Szoftvertechnológia

Szigorlati tételek (2005)

Összeállította:
Együd Dániel
Kovács László

Bár igyekeztünk alapos munkát végezni, helyenként lehet, hogy előfordulnak elírások, és egyéb hibák, ezekért felelősséget nem vállalunk, de ha erre valaki felhívja a figyelmünk, igyekszünk rövid időn belül kijavítani.

A teljes anyagot mindenki fellelheti Vető Tanár Úr honlapján (http://digitus.itk.ppke.hu/~veto).

Mindenkinek sikeres szigorlatot kívánunk.

Együd Dániel, Kovács László
Budapest, 2005. november

1. A szoftvertervezés és más tervezési folyamatok összehasonlítása

A számítógép alapú rendszerek tervezése a rendszerek valamennyi aspektusával foglalkozik: a folyamatok tervezése, hardver-, szoftver-, architektúra specifikációja, valamint tervezése, továbbá az integráció megoldása.
A szoftvertervezés a fentiek szerint a rendszertervezés része.

A számítógép-tudomány az elméletekkel és módszerekkel foglalkozik.
A szoftver technológia a használható szoftver előállításának és használatának gyakorlati kérdéseire keres választ.

- Gyakorlati, mert meg kell oldani olyan problémákat is, amelyekre a tudomány még nem találta meg a helyes megoldást

A számítógép-tudomány elméletei jelenleg még nem mindig alkalmazhatók a bonyolult, valódi feladatok megoldására.

- Nem megfogható: A szoftvertervezés elsősorban abban tér el más tervezési folyamatoktól, hogy ennek termékét nem lehet kézzel megfogni, illetve a hétköznapi értelemben vett mérés sem alkalmazható rá (nem lehet látni, ha csúszás van).
  Különféle módszereket kellett kidolgozni annak érdekében, hogy a szoftverek tervezése ellenőrizhető, illetve nyomon követhető legyen. Ilyenek például:
  - felhasználói felület mérése (felhasználók bevonásával)
  - komponensek mérése (gyorsaság, megbízhatóság, optimalitás, stb)
  - azonosítók hossza
  - stb
- Nincsenek szabványos szoftverfolyamatok: még nincsenek kiforrott módszerek, nem tudjuk nagy biztonsággal megmondani, mikor fog egy szoftverfolyamat hibákba ütközni.
- A nagy szoftverprojektek gyakran teljesen újak: Sokszor előfordul,hogy olyan rendszert kell kidolgozni, amilyen addig még nem létezett, ezért csak a hasonló projektekből lehet tapasztalatokat gyűjteni.

2. A szoftverfolyamat fázisai és modelljei, szoftverfejlesztési stratégiák

Fázisai:
- Szoftverspecifikáció: a szoftver feladatainak és a megszorításoknak specifikációja
- Szoftverfejlesztés: a szoftver rendszer elkészítése
- Szoftvervalidáció: annak bizonyítása, hogy az elkészített rendszer a követelményeknek megfelelően működik
- Szoftverevolúció: a szoftver továbbfejlesztése a változó igényeknek megfelelően

A modell a szoftverfolyamat egy adott nézőpontból való leegyszerűsítése.

Példák a szoftverfolyamat-modellek típusaira:
- Munkafolyamat modell: a tevékenységek sora
- Adatfolyam modell: az információáramlás modellje
- Szerepkör/cselekvés modell: a szoftverfolyamat résztvevőinek szerepe és tevékenységei

Általános szoftverfolyamat modellek:
- Vízesés:
  fázisai:
  1. Követelmények elemzése és meghatározása
  2. Rendszer- és szoftver tervezés
  3. Implementáció és egységteszt
  4. Integráció és rendszerteszt
  5. Működtetés és karbantartás

  Előnyei:
  - Jól áttekinthető és követhető fejlesztési projekt folyamatot eredményez.
  - A folyamat termékei szerződésekkel könnyen lefedhetők (specifikációs és tervezési dokumentumok, program(ok), stb.)

  - Hátrányai:
    - Egymástól elkülönült fázisokra osztja a projektet (költséges egy korábbi fázishoz visszatérni pl. specifikációs, vagy tervezési hiba esetén).
    - Csak a projekt végén, az átadáskor (a működtetés első lépésekor) derülnek ki a specifikációs hibák).
    - Nem képes rugalmasan alkalmazkodni a felhasználói igények változásaihoz.
    - Csak a követelmények pontos ismeretében alkalmazható.

- Evolúciós fejlesztés:
  Az alapgondolat: ki kell dolgozni egy kezdeti implementációt, amelyet a felhasználó véleményezhet, és azt finomítani az elfogadásig.

  Feltáró fejlesztés:
  A követelmények feltárása lépésenként, a megrendelővel együttműködve történik, folyamatosan kiegészítve a rendszert az új funkciókkal, részekkel.

  Eldobható prototípus
  „Deszkamodellek” készítése és átadása az ügyfélnek, a követelmények pontosabb feltárása érdekében.

Előnyök:
- Kis-, vagy közepes interaktív rendszerek, vagy nagy rendszerek felhasználói interfészének fejlesztésére alkalmas
- Rövid életciklusú rendszerek esetén előnyös

- Hátrányok:
- A projekt előrehaladása nem követhető
- A rendszerek struktúrájával nem foglalkozik
- Speciális eszközöket és ismereteket igényel (pl. gyors modellező eszközök alkalmazása)

- Formális transzformációk:
  - A vízesés modellhez hasonló, de a fejlesztés formális matematikai eszközökkel állítja elő a futtatható programot a rendszerspecifikáció matematikai modelljéből, több transzformációs lépésen keresztül.
  - Minden transzformáció során, lépésenként kell végrehajtani a tesztelést

  Előnyök:
  - Kritikus rendszerek esetén, ahol kulcskérdés a biztonságosság, megbízhatóság, vagy védelem.
  - A transzformáció és a bizonyítás részben automatizálható.

  Hátrányok:
  - Speciális szakértelmet igényel.
  - Egy rendszer kölcsönhatásait (pl. felhasználói interfész) nehéz formálisan specifikálni.
  - Komplex, nagy rendszereknél ez a módszer sem eredményez jobb minőséget, vagy költségmegtakarítást

- Integráció újrafelhasználható komponensekből:
  - Már létező, újrafelhasználható szoftver-komponensek egységes szerkezetbe való integrálása. (komponens alapú rendszerfejlesztés)
  - „Polcról levehető” (COTS – Commercial off the Shelf) termékek felhasználása.
  - Gyakran beépül a korábban ismertetett folyamatokba.

  A folyamat lépései:
  - Követelmények meghatározása
  - Komponens elemzés
  - Követelmények módosítása
  - Rendszertervezés újrafelhasználással
  - Fejlesztés és integráció
  - Rendszervalidáció

Egy fejlesztési projekt során a rendszerkövetelmények változhatnak, ezért egyes lépéseket meg kell ismételni.
Az iteráció az általános folyamatmodellek bármelyikével alkalmazható (hibrid modellek).
Ez gyakran ellentmond a projektszervezési-, szerződéskötési elvárásoknak, amelyek jól definiált termékeket (dokumentum, vagy program) követelnek.
Két hibrid modellt ismertetünk:
- Inkrementális fejlesztés
- Spirális fejlesztés

- Inkrementális fejlesztés:
  - A szoftverfejlesztést több, kisebb vízesés modellre bontja, mindegyik a funkciók egy

meghatározott részét valósítja meg.
- Az első lépésekben a legfontosabb felhasználói követelményeket elégíti ki.
- Amikor egy (rész)fázisban a fejlesztés megindul, az arra vonatkozó követelményeket befagyasztja.
- Új fogalom: extrém programozás. (kis funkcionalitással rendelkező inkrementumok)

Előnyök:
- A felhasználó minden inkrementum átadásakor újabb, használható funkciókat kap.
- Az első inkrementumok prototípusként szolgál-hatnak, és segítik a későbbi inkrementumok követelményeinek meghatározását.
- A teljes projekt meghiúsulásának alacsonyabb a kockázata.
- A legfontosabb szolgáltatások hamarabb leszállíthatók, így azokat alaposabban tesztelik.

Hátrányok:
- Nagy rendszerekben sok olyan alapvető funkció van, amelyek nélkül rendszerszolgáltatás nem működhet. Nehéz olyan inkrementumokat definiálni, amelyek ezek nélkül is használható szolgáltatást nyújtanak.
- A már leszállított inkrementumokban nagyon nehéz módosítani, ha az inkrementumok kölcsönhatása miatt szükségessé válik.

- Spirális fejlesztés:
  - A szoftver folyamatot nem tevékenységek és esetleges visszalépések sorozataként, hanem spirálisként reprezentálja.
  - A spirál minden köre a szoftverfolyamat egy fázisa.
  - A spirálban nincsenek fix lépések, mint pl. specifikáció, vagy tervezés, minden körben kidolgoznak egy új, az előzőnél teljesebb prototípust, amelynek „utolsó” verzióját részletes követelmény specifikációnak tekintik.
  - Ennek alapján készül el a „végleges” termék.

3. A CASE eszközök, rendszerek és osztályozásuk

A CASE-eszközök olyan szoftver rendszerek, amelyek a szoftver folyamat tevékenységeit támogatják, automatizálják. Általában konkrét módszerhez kapcsolódnak.

Magas szintű CASE-eszközök
- A szoftverfolyamat kezdeti lépéseit támogatják:
  elemzés, tervezés, modellezés, rendszer dokumentálás, jelentés-készítés, stb.

Alacsony szintű CASE-eszközök
- A szoftverfejlesztés későbbi tevékenységeit támogatják, mint kódszerkesztés (kódgenerálás!), kódelemzés, nyomkövetés, tesztelés, stb.

A CASE eszközök az alábbiak szerint osztályozhatók:
- Funkcionális szempontból: Aszerint, hogy az eszköz milyen funkciót lát el.
- A folyamat szempontjából: A támogatott folyamat tevékenységei szempontjából.
- Integráltságuk szempontjából: Aszerint, ahogyan egy vagy több folyamat tevékenységet támogatnak.

Funkcionális osztályozás:

Eszköztípus	**Példa**
Tervezőeszközök	PERT eszközök, becslési eszközök, táblázatkezelők
Szerkesztő eszközök	Diagramszerkesztők, szövegszerkesztők
Változáskezelő eszközök	Követelmény követhetőségi eszközök, változtatásvezérlő rendszerek
Konfigurációkezelő eszközök	Verziókezelő rendszerek, rendszerépítő eszközök
Prototípuskészítő eszközök	Magas szintű programnyelvek, felhasználói interfész generátorok, munkafolyamat tervező eszközök
Módszertámogató eszközök	Tervszerkesztők, adatszótárak, kódgenerátorok
Nyelvi feldolgozó eszközök	Fordítók, értelmezők
Programelemző eszközök	Keresztreferencia generátorok, statikus elemzők, dinamikus elemzők
Tesztelő eszközök	Tesztadat generátorok, állomány összehasonlítók
Nyomkövető eszközök	Interaktív nyomkövető és belövő eszközök
Dokumentációs eszközök	Arculattervező programok, képszerkesztők
Újratervezési eszközök	Kereszthivatkozási rendszerek, program újrastrukturáló rendszerek

Integráltság:
- Eszközök: Az egyes folyamatlépéseket támogatják, mint a terv konziszetciájának ellenőrzése, program fordítás, teszteredmények összehasonlítása, stb.
- Eszközkészletek (workbench): A szoftverfolyamat egyes fázisait támogatják, mint pl. specifikáció, vagy tervezés. Általában több, egymással együttműködő eszközből állnak.
- Környezetek: A szoftverfolyamat több fontos, vagy valamennyi részét támogatják. Legtöbbször több, integrált eszközkészletből állnak.

Szoftverfolyamat szempontjából:

	Specifikáció	Tervezés	Implementáció	Verifikáció validáció
Tervezőeszközök	●	●	●	●
Szerkesztő eszközök	●	●	●	●
Dokumentációs eszközök	●	●	●	●
Változáskezelő eszközök	●	●	●	●
Konfigurációkezelő eszközök		●	●	
Prototípus-készítő eszközök	●			●
Módszertámogató eszközök	●	●		
Nyelvi feldolgozó eszközök		●	●	
Programelemző eszközök			●	●
Nyomkövető eszközök			●	●
Tesztelő eszközök			●	●
Újratervezési eszközök			●	

4. A szoftverfejlesztési projekt jellemzői, ütemezése, a projekt kockázatai, a kockázatok kezelése

- A projektvezetés feladata, hogy a szoftver a tervezett ütemezés szerint, határidőre, a követelményeknek megfelelően készüljön el.
- A projekt menedzselésére azért van szükség, mert a szoftverfejlesztés mindig kötött pénzügyi és megszabott időkeretek között folyik, amelyeket a megrendelő, vagy a fejlesztő szervezet jelöl ki.

A menedzsment feladatai:
- Ajánlatkészítés
- Projekt tervezés és ütemezés
- Projekt költségvetés tervezése
- A projekt felügyelete, ellenőrzése
- Személyek kiválasztása és vezetése
- Beszámolók és ismertetők készítése

Személyek kiválasztása:
Általában nem biztosítható, hogy a legjobban felkészült, tapasztalt szakemberek álljanak rendelkezésre, mert:
- A projekt költségvetése nem engedi meg a legjobban fizetett szakemberek bevonását
- Az adott projekt teljesen új, így nincsenek még szakemberei
- A kevésbé tapasztalt szakemberek képzése lehet a vezetés feladata

A projekt vezetésének ilyen megkötéseket is figyelembe kell vennie, miközben nemzetközi méretekben is hiány van képzett, tapasztalt IT szakemberekben

A projekt tervezése
- A projektvezetés leginkább időigényes feladata.
- Folyamatos tevékenység. A koncepció kidolgozásától a rendszer átadásáig tart. Az előrehaladást folyamatosan követni kell, projekttervet rendszeresen felül kell vizsgálni és át kell dolgozni a változó állapotnak megfelelően.
- Gyakran többféle, különböző projekttervet kell kidolgozni (pl. előre várható vészhelyzetekre felkészülve)
- „A projektvezető sohasem lehet optimista”

A projektterv típusai:

Típus	**Jellemzők**
Minőségi terv	Meghatározza a projektben használandó minőségbiztosítási eljárásokat és szabványokat
Validációs terv	Meghatározza a rendszer validációja során használandó módszereket, erőforrásokat, ütemezést
Konfiguráció-kezelési terv	Leírja a konfiguráció-kezelés eljárásait és struktúráját
Karbantartási terv	A karbantartás követelményeinek, költségeinek terve
Munkaerő-fejlesztési terv	Terv a projekten dolgozó csapat szaktudásának, tapasztalatainak fejlesztésére

A projektterv felépítése:
- Bevezetés
- A projekt szervezete
- Kockázatelemzés
- Szükséges hardver és szoftver erőforrások

- A munka felosztása
- A projekt ütemterve
- A projekt előrehaladásának megfigyelésére használandó technikák, a készítendő jelentések, beszámolók meghatározása

A tevékenységek megszervezése:
A tevékenységeket úgy kell szervezni, hogy a projektvezetés kézzelfogható részeredmények sorával követhesse az előrehaladást:
- ***Mérföldkő:*** az egyes tevékenységek mérhető befejezése
- ***Termék:*** a megrendelőnek leszállítandó (rész)eredmény
- A mérföldkövek kijelölésére a vízesés modell a legalkalmasabb

Mérföldkövek:

Tevékenység	Mérföldkő
Megvalósíthatósági vizsgálat	Megvalósíthatósági tanulmány
Követelményelemzés	Felhasználói követelmények
Prototípusfejlesztés	Értékelési jelentés
Tervtanulmány	Architektúra terv
Követelményspecifikáció	Rendszerkövetelmények

A projekt ütemezése:
- A projektet tevékenységekre kell bontani, mindegyikhez meghatározva a végrehajtáshoz szükséges időt és erőforrásokat.
- Egyes tevékenységeket párhuzamosan is el lehet végezni.
- A tevékenységek közti függőségeket minimálisra kell csökkenteni, hogy az egyik késlekedése ne befolyásolja a többi előrehaladását.
- A jó (végrehajtható) ütemezés a projektvezető tapasztalatán és megérzésein múlik (váratlan problémák betervezése)

Az ütemezés nehézségei:
- A munka közben fellépő nehézségeket nem könnyű előre megbecsülni, így a fejlesztés költségei is csak közelítőleg becsülhetők.
- A termelékenység nem egyenesen arányos a résztvevők számával.
- Egy késedelmes projekt nem gyorsítható fel újabb emberek bevonásával.
- A váratlan események mindig bekövetkeznek, ezért ráhagyással kell tervezni.

Oszlopdiagramok és tevékenységhálók:
- A grafikus ábrázolás érhetőbbé teszi a projekt ütemezését és a tevékenységek közti összefüggéseket.
- A projektet egy-két hetes tevékenységekre érdemes bontani.
- A tevékenységháló bemutatja a függőségeket és a kritikus utat,
- Az oszlopdiagram az ütemezést az idő függvényében ábrázolja és bemutatja a felelősöket.

Kockázatkezelés:
- A kockázatkezelés a lehetséges kockázati tényezők azonosítását és a projektre gyakorolt hatásuk minimalizálására vonatkozó tervek készítését jelenti.
- A kockázat típusai:
  - **Projektkockázat**: A projekt ütemtervét, vagy az erőforrásokat veszélyezteti,

- **Termékkockázat**: A szoftver minőségét, vagy teljesítményét veszélyezteti
- **Üzleti kockázat**: A szoftver beszerzését, vagy fejlesztését végző szervezetet veszélyezteti

A kockázatok típusai:

Kockázat	Típus	Leírás
Munkaerő változása	Projekt	Egy, vagy több tapasztalt munkaerő elhagyja a projektet
Vezetőség változása	Projekt	A projekt vezetőségének változása miatt megváltoznak a prioritások
Hw/sw hiánya	Projekt	A szükséges hw vagy alapszoftver nem áll rendelkezésre
Követelmények változása	Projekt Termék	A követelmények váratlan, nagymértékű megváltozása
Specifikáció késése	Projekt Termék	A szükséges interfészek specifikációja nem áll időben rendelkezésre
Méret alábecslése	Projekt Termék	A tervezéskor alulbecsülték a rendszer méretét
CASE eszköz alkalmatlansága	Termék	A projektet támogató CASE eszköz nem alkalmas a feladatra
Technológia megváltozása	Üzleti	A rendszer alapjául szolgáló technológia elavul
Termékverseny	Üzleti	A rendszer elkészülte előtt megjelenik egy versenyképes termék

A kockázatkezelés folyamata:

- A kockázat azonosítása: A lehetséges projekt-, termék-, és üzleti kockázatok azonosítása.
- A kockázat elemzése: A kockázatok valószínűségének és hatásainak becslése.
- A kockázat tervezése: Tervek készítése a kockázatok elkerülése, illetve hatásuk csökkentése érdekében.
- A kockázatok figyelése: A kockázatok és hatásuk figyelése a teljes projekt során, a tervek frissítése

A kockázatok típusai:

Kockázat típus	Lehetséges kockázat
Technológiai	Az adatbázis-kezelő rendszer nem képes annyi tranzakciót feldolgozni másodpercenként, mint várták. Az újrafelhasznált komponensek a funkcionalitást befolyásoló hibákat tartalmaznak.
Emberi	Nem lehet a csapatot a kívánt képzettségű emberekből összeállítani. A kulcsemberek megbetegszenek a kritikus időszakban.
Szervezeti	A szervezet menet közben megváltozik, új emberek felelősek a projektért. A szervezet pénzügyi helyzete miatt csökkenteni kell a projekt költségvetését.
Eszközök	A CASE eszköz kódgenerátora nem képes effektív kódot létrehozni. A CASE eszközt nem lehet integrálni.
Specifikáció	A szükséges interfészek specifikációja nem áll időben rendelkezésre
Követelmények	A rendszerkövetelményekben olyan nagy változás következik be, hogy a rendszert újra kell tervezni.
Becslés	Alulbecsülték a fejlesztéshez szükséges időt. Alulbecsülték a szoftver méretét.

Kockázatok tervezése:
Vegyünk figyelembe minden kockázatot és dolgozzunk ki stratégiát a kezelésére. Lehetséges stratégiák:

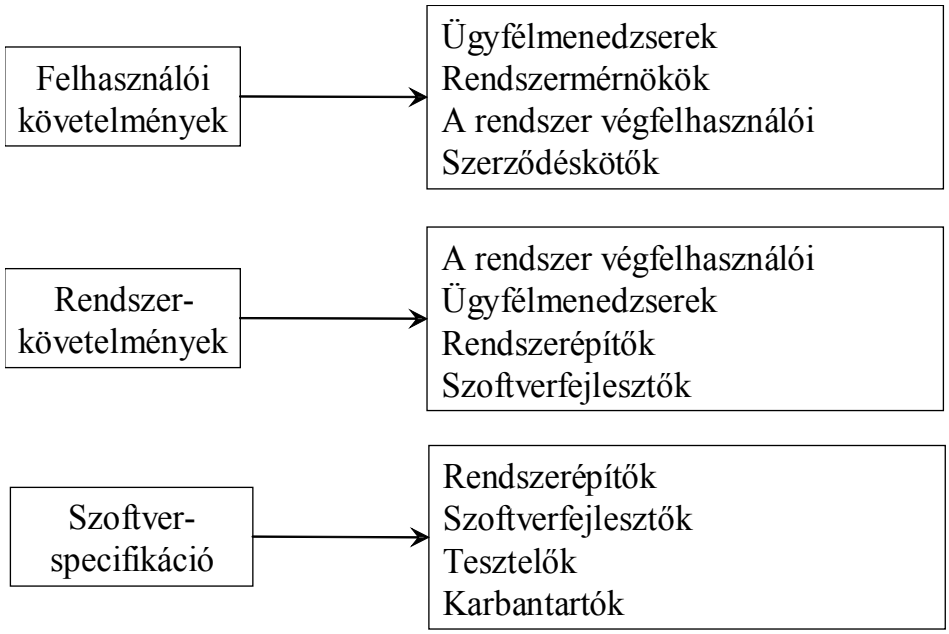
- A kockázatkerülés stratégiája: Csökkenti a kockázat valószínűségét.
- Minimalizáló stratégia: Csökkenti a kockázat hatását a projektre, vagy termékre.
- Biztonsági tervek: Előre elkészített akciótervek arra az esetre, ha a kockázat bekövetkezik

5. A szoftverkövetelmények típusai, a követelmények elemzése

I. csoportosítás

- Felhasználói követelmények: A rendszer szolgáltatásainak közérthető leírása, diagramokkal, táblázatokkal, ábrákkal, a felhasználó számára.
- Rendszerkövetelmények: Strukturált dokumentum a rendszer szolgáltatásainak részletes leírásával. Ez a szerződés alapja.
- Szoftverterv specifikáció: A szoftver részletes leírása a fejlesztők számára

Specifikációk olvasói:



A felhasználói követelmények:

- A felhasználói követelményeket úgy kell megfogalmazni, hogy az informatikában járatlan felhasználó is megértse.
- Ezért itt nem célszerű modelleket alkalmazni, hanem természetes nyelven, űrlapokkal és diagramokkal kell a felhasználói követelményeket érthetővé tenni.
- A természetes nyelv alkalmazásának nehézségei:
  - Az egyértelműség és pontosság hiánya
  - A követelmények keveredése
  - A követelmények ötvöződése

Rendszerkövetelmények:

- A rendszerkövetelmények a felhasználói követelmények részletesebb leírását adják.
- A rendszertervezés alapjául szolgálnak, tartalmazhatják a rendszer modelljeit.
- Sokszor a szerződéshez csatolják, ezért a rendszer teljes és konzisztens meghatározását kell tartalmazniuk.
- A rendszerkövetelmények leírják, hogy a rendszernek mit kell elvégeznie, a tervek azt határozzák meg, hogy hogyan tegye.

Szoftverterv specifikáció:

- A követelmények dokumentuma (System Requirements Specification – SRS) írja le, hogy mit várnak a tervezett rendszertől, vagyis mit kell megvalósítania a tervezőknek.

- A dokumentumnak nem a rendszer tervét kell tartalmaznia, hanem a követelmények definícióit és specifikációját vagyis azt, hogy **mit kell tennie a rendszernek és nem azt, hogy hogyan**.

II. csoportosítás
- Funkcionális: A rendszer által nyújtandó szolgáltatások leírása: hogyan reagáljon a rendszer az egyes bemenetekre, illetve mit tegyen egyes helyzetekben.
- Nem funkcionális: A rendszer funkcióira és szolgáltatásaira vonatkozó megszorítások, időbeli korlátok, a fejlesztési folyamatra vonatkozó korlátozások, szabványok.
- Szakterületi: A rendszer alkalmazási szakterületéről származó funkcionális, vagy nem-funkcionális követelmények (mint pl. törvényi szabályozások, stb.)

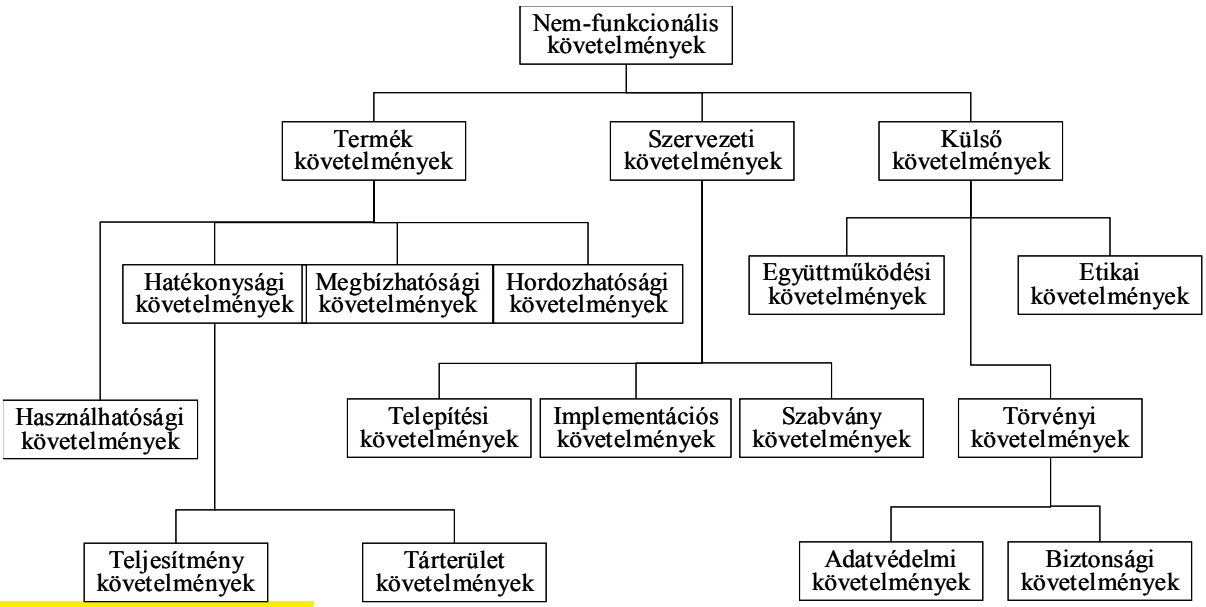
Funkcionális követelmények:
- A rendszer funkcióit, illetve szolgáltatásait tartalmazzák.
- A szoftver típusától, a várható felhasználástól és a felhasználóktól függenek.
- A felhasználói funkcionális követelmények általánosan írják le, hogy mit kell elvégeznie a rendszernek.
- A funkcionális rendszerkövetelmények részletesen írják le az egyes funkciók bemeneteit, kimeneteit, a kivételeket, stb.

Követelmények precíz meghatározása:
- A követelményeket pontosan kell leírni, az olvasó számára egyértelműen.
- A pontatlan követelmény specifikáció félreértéseket eredményez.
- Például:
  - A felhasználó követelménye: „Megfelelő megjelenítőt kell biztosítani minden dokumentum típushoz”
  - A fejlesztő értelmezheti úgy, hogy csak egy szöveges megjelenítőre van szükség, így az összetett dokumentumokat nem lehet olvasni.

A követelmények teljessége, konzisztenciája:
- A követelményeket elvileg mindenre kiterjedően, és ellentmondásmentesen kell leírni.
- A teljesség azt jelenti, hogy a felhasználó által igényelt összes szolgáltatásnak szerepelnie kell.
- A leírás konzisztens, ha nincs konfliktus, vagy ellentmondás a rendszer szolgáltatásai között.
- A gyakorlatban ez a nagyméretű, komplex rendszereknél megoldhatatlan.

Nem funkcionális követelmények:
- A rendszerfunkciókon kívüli követelmények, mint a megbízhatóság, válaszidő, tárigények, vagy az I/O eszközök tulajdonságaira, az interfészek adatformátumaira, stb. vonatkozó megszorítások.
- Ide tartoznak a fejlesztés módszereire, a minőség-ellenőrzésre, a fejlesztőeszközökre (CASE) vonatkozó követelmények, vagy a rendszeren kívüli (pl. jogi) megkötések is.
- Még a funkcionális követelményeknél is kritikusabbak lehetnek (pl. repülőgép irányítás – megbízhatóság)

A nem funkcionális követelmények osztályozása:
- A termékre vonatkozó követelmények: A termék viselkedését határozzák meg (pl. sebesség, megbízhatóság, hordozhatóság, stb.)
- Szervezeti követelmények: A megrendelő és a fejlesztő szervezete által támasztott szabályzatok és ügyrendek követelményei (módszertan, programozási nyelv, stb.)
- Külső követelmények: A rendszeren és a fejlesztésen kívüli követelmények. (együttműködés más rendszerekkel, jogszabályi, etikai, stb.)

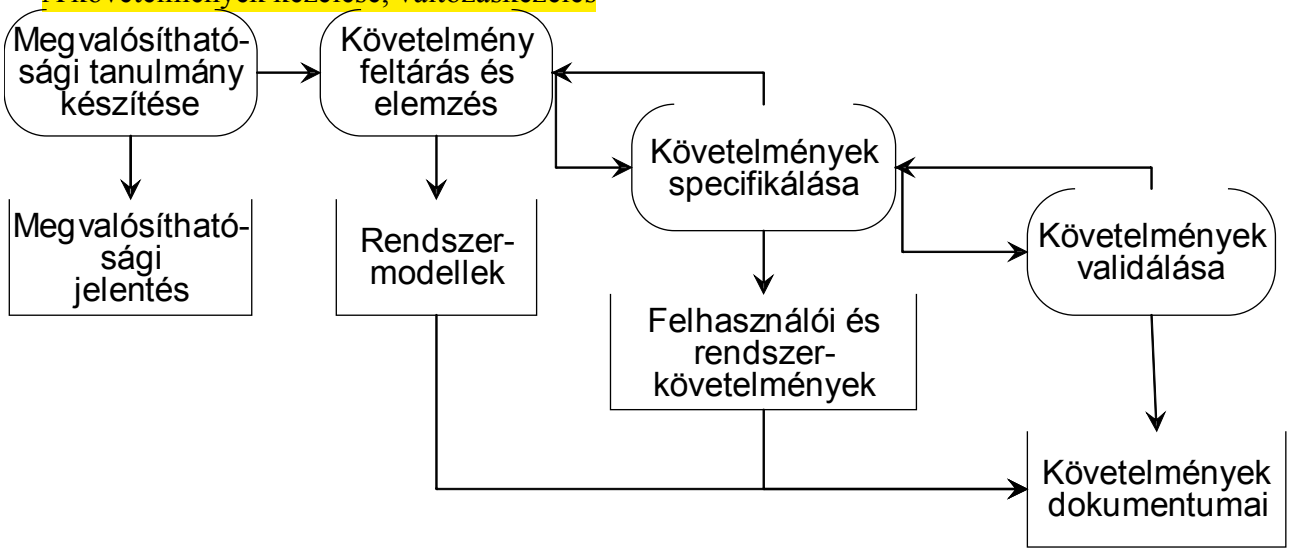


Szakterületi követelmények:

- A rendszer alkalmazásának szakterületéből származó követelmények. Ha a rendszer ezeket nem elégíti ki, használhatatlan lesz.
- A szakterületi követelmények jelenthetnek új funkcionális követelményeket, vagy megszorításokat a meglévő követelményekhez.
- Nehézségek:
  - A szakterületi követelményeket az alkalmazási terület nyelvén fogalmazzák meg, ezt a fejlesztők nem értik.
  - A szakterület specialistái gyakran kézenfekvőnek tartanak, és nem fogalmaznak meg olyan követelményeket, amelyekre a fejlesztők nem is gondolnak.

A követelménytervezés folyamatai:

- Megvalósíthatósági tanulmány
- A követelmények feltárása és elemzése
- A követelmények validálása
- A követelmények kezelése, változáskezelés



Megvalósíthatósági tanulmány:

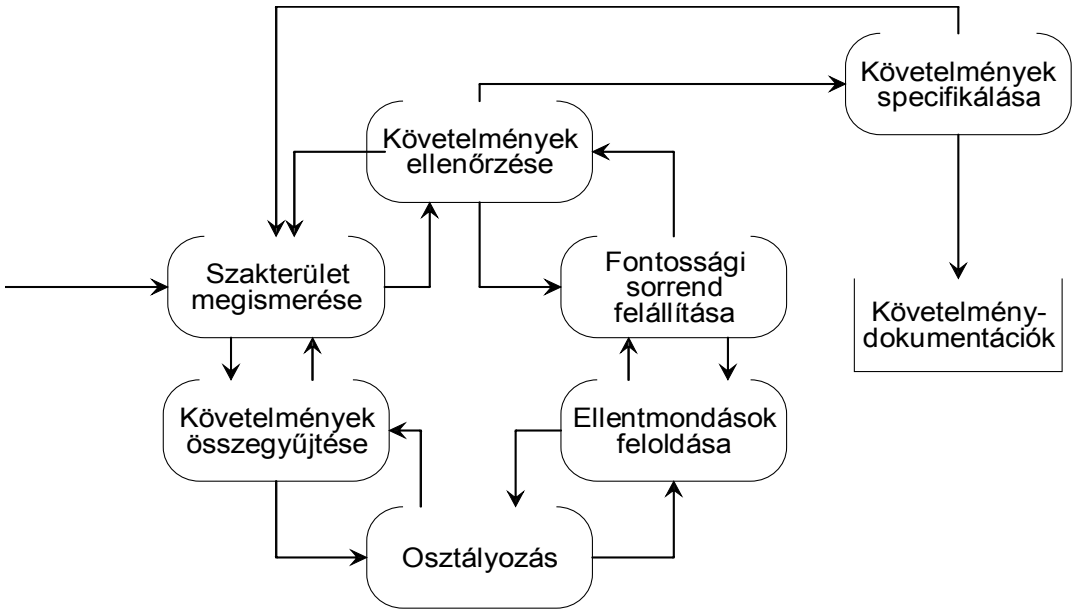
- Feladata: megalapozni azt a döntést, hogy érdemes-e a tervezett rendszert megvalósítani.

- A tanulmány az alábbi kérdésekre ad választ:
  - Más, hasonló szervezeteknél milyen megoldásokat alkalmaztak hasonló feladatokra.
  - Mennyiben támogatja a rendszer a megrendelő általános célkitűzéseit.
  - Megvalósítható-e a rendszer a tervezett költségen belül, az adott technológiával, a kívánt határidőre.
  - Integrálható-e a rendszer más, már meglévő rendszerekkel.

Megvalósítósági tanulmány készítése:
- *Az alábbi információk felmérésén, összegyűjtésén alapul*
  - Milyen problémák vannak a jelenlegi folyamatokkal, és a tervezett rendszer hogyan segít ezeket feloldani.
  - Hogyan járul hozzá a rendszer a megrendelő üzleti céljaihoz.
  - Milyen folyamatokat kell támogatnia a rendszernek és melyeket nem.
  - Mi történne, ha a rendszert nem valósítanák meg.
  - Hogyan illeszkedik a rendszer a meglévő rendszerekhez.
- A tanulmány ajánlásokat tartalmaz a rendszer funkcióira és a megvalósítás módjára.

Követelmények feltárása és elemzése:
- Az informatikusok interjúkat készítenek a megrendelő kulcsembereivel (szakterületi képviselők, vezetők, és majdani felhasználók), hogy felderítsék, milyen szolgáltatásokat kell biztosítania a rendszernek.
- A követelmények feltárása bonyolult, mert:
  - A kulcsemberek gyakran nem tudják, hogy mit várhatnak és várnak egy számítógépes rendszertől.
  - A kulcsemberek a saját szakterületük fogalmait használják, a követelménytervezőknek ezeket kell megérteniük.
  - Az egyes szakterületeknek különböző elvárásai vannak.
  - Egyes kulcsfigurák a saját pozíciójuk erősítésére akarják felhasználni az új rendszert.
  - A környezet változása folyamatosan módosítja a követelményeket, a változásokat követni kell.

A feltárás és elemzés folyamata:



Nézőpont-orientált feltárás:

- Nagy rendszerek különböző felhasználói különböző nézőpontból látják a rendszer szolgáltatásait, eltérő követelményeik vannak, amelyek gyakran átfedik egymást.
- A rendszerkövetelmények feltárását és elemzését tehát több perspektívából kell végezni, nincs egyetlen helyes út a probléma megközelítésére.
- A nézőpont-orientált szemlélet felismeri a különböző perspektívákat és segít a különböző kulcsszereplők egymásnak ellentmondó követelményeinek felfedésében.
- A nézőpontok kívülről tekintik a rendszert, így strukturálják a követelmények feltárását.
- Főleg interaktív rendszerekhez alkalmas.

Nézőpontok típusai:
- Adatforrás vagy adatnyelő: Az adatok előállításának és feldolgozásának nézőpontjai. Az elemzés kiterjed az összes adatforrásra és adatnyelőre, továbbá a feldolgozások azonosítására és vizsgálatára.
- Reprezentációs eszközkészlet: A különböző típusú rendszermodellek nézőpontjai. Ezek összehasonlításából kitűnik, melyek azok a követelmények, amelyeket az egyes rendszermodellek hibásan értelmeznek.
- A szolgáltatások fogadója: A rendszer szolgáltatásait felhasználó (ember, másik rendszer, stb.) nézőpontjai.

Módszerek a követelményelemzéshez:
- Céljuk a rendszerkövetelmények strukturált feltárásának, elemzésének, tervezésének támogatása.
- Típusok:
  - Nézőpont orientált módszerek
    (pl. VORD – Viewpoint-Oriented Requirements Definition)
  - Forgatókönyvek:
    - Esemény forgatókönyvek: Leírják, hogyan használják a rendszert a gyakorlatban.
    - Használati esetek: (Az UML-nél foglalkozunk vele).
  - Etnográfia: A rendszerek társadalmi, szervezeti környezete, az emberek munkavégzési módja felől közelíti a rendszerkövetelményeket. Gyakran kiegészül a prototípus-készítéssel.

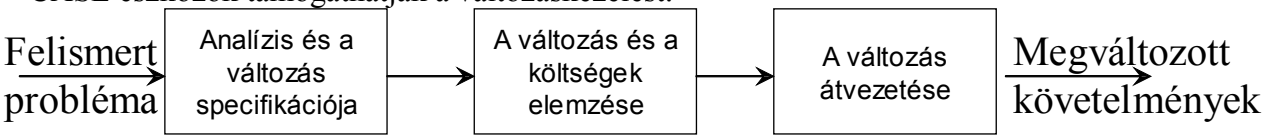
Követelmények validálása:
- Feladata annak igazolása, hogy a követelmények a megrendelő kívánságainak megfelelő rendszert definiálják.
- A hibás, vagy hiányos követelmények nagy veszteségeket okoznak, ezért a validáció igen fontos:
  - A hibás követelmények javítása a rendszer átadása után százszor annyiba kerül, mint egy implementációs (pl. programozási) hiba kijavítása.

Követelmények ellenőrzése:
- Az érvényesség ellenőrzése: A felhasználó által előre nem látott funkciók feltárása.
- Az ellentmondás-mentesség ellenőrzése: Ellentmondó megszorítások, vagy rendszerfunkciók kiszűrése.
- A teljesség ellenőrzése: Annak ellenőrzése, hogy a dokumentum a felhasználó által kért összes funkciót tartalmazza-e.
- A megvalósíthatóság ellenőrzése: Az elképzelt rendszer megvalósítható-e a rendelkezésre álló technológiával, a tervezett idő alatt, az adott költséggel.
- Az igazolhatóság ellenőrzése: A rendszerkövetelmények dokumentumai alkalmasak-e arra, hogy utólag igazolják, az átadott rendszer teljesíti a követelményeket

Követelmények kezelése:
- A követelmények kezelése a követelmények változásának követésére, kézbentartására szolgáló folyamat.

- A követelmények soha nem lehetnek teljesek és konzisztensek.
  - A szoftverfolyamat során új követelmények merülnek fel, ahogy az üzleti környezet változik, és a feladat megértésében előbbre jutunk.
  - A különböző nézőpontok különböző követelményeket támasztanak a rendszerrel szemben, amelyek gyakran ellentmondanak egymásnak.

Követelmények változásának kezelése:
- A követelménykezelés során fel kell készülni a követelmények változására. Ehhez szükség van:
  - A követelmények egyedi azonosítására.
  - A változáskezelés folyamatának kidolgozására.
  - A követelmények és az egyes közötti összefüggések változásának követésére.
  - A változások és hatásuk elemzésére.
- CASE eszközök támogathatják a változáskezelést.



6. Tervezés újrafelhasználással, komponensek felhasználása

Szoftver újrafelhasználása:
- A legtöbb mérnöki tervezési tevékenység komponensek újrafelhasználásán alapul. A terveket más rendszerekben már kipróbált, szabványos, kisebb-nagyobb komponens újrafelhasználására alapozzák (csavaroktól a hajtóművekig).
- A szoftverfejlesztés hagyományosan az eredeti fejlesztésen alapul, de a minőség javítása, a költségek, és a fejlesztési idő csökkentése érdekében mindinkább előtérbe kerül a szoftver komponensek újrafelhasználása.
- Ehhez olyan tervezési módszereket kell alkalmazni, amely a szisztematikus újrafelhasználáson alapul.

Újrafelhasználáson alapuló szoftverfejlesztés:
- Alkalmazási rendszerek újrafelhasználása: Teljes alkalmazási rendszerek újrafelhasználása:
  - Beépítve más rendszerekbe, vagy
  - Speciális felhasználói igényeket kiszolgáló alkalmazás-családok kifejlesztése.
- Komponensek újrafelhasználása: Különböző méretű (objektum – alrendszer) komponensek beépítése új rendszerekbe.(pl. driverek, interfész modulok, stb.)
- Függvények újrafelhasználása: Egyszerű, jól definiált tevékenységet végző komponensek újrafelhasználása. (pl. szabványos könyvtárak)

Az újrafelhasználás előnyei:
- Javuló megbízhatóság: A komponenseket már több működő rendszerben kipróbálták.
- Alacsonyabb projektkockázat: komponensek ára és adaptálási költsége pontosabban tervezhető.
- A szaktudás jobb kihasználása: A speciális szaktudás a komponensben testesül meg, nem szükséges minden projekthez külön alkalmazni.
- Szabványosság: A szabványoknak való megfelelést a komponensek garantálják (interfészek, kommunikációs és GUI szabványok)
- Gyorsabb fejlesztés: Egy rendszer kifejlesztése gyorsabb, ha kevesebb eredeti fejlesztést igényel.

Az újrafelhasználás hátrányai:
- Növekvő karbantartási költségek: A komponens forráskódja és tervezési dokumentációja hiányában növekszik a karbantartás költsége.
- Az eszköztámogatás hiánya: A CASE eszközök nem támogatják az újrafelhasználást.
- A „nem mi találtuk ki” jelenség: Egy teljes rendszer kidolgozása nagyobb szakmai kihívás.
- A komponenskönyvtárak karbantartása: Sokba kerül a komponenskönyvtárak feltöltése és folyamatos karbantartása.
- Az újrafelhasználható komponensek megtalálása és adaptálása: Még nem fejlődtek ki a komponensek megtalálását és adaptálását segítő általános technikák.

Kritikus követelmények:
- Meg kell találni a megfelelő újrafelhasználható komponenseket. Ehhez katalógusokra és nyilvántartásokra, alkalmas kereső mechanizmusokra van szükség.
- Az újrafelhasználónak bíznia kell abban, hogy a komponens a leírtaknak megfelelően és megbízhatóan működik.
- A komponenseknek olyan dokumentációval kell rendelkezniük, amely érthető, teljes, aktuális és hivatkozik a korábbi felhasználásokra is (referenciák).

Újrafelhasználás programgenerátorral:
- A generátor alapú újrafelhasználás akkor lehetséges, ha egy programgenerátor tartalmazza egy

szakterület alapvető ismeretanyagát. (pl. adatfeldolgozás)
- Az ilyen programgenerátorok tartalmazzák a szabványos algoritmusokat és függvényeket, és ezek paraméterezését követően a generátor automatikusan előállítja a programot.
- A szakterületre kidolgozott nyelven, vagy újabban grafikus eszközökkel lehet elkészíteni a rendszer modelljét.

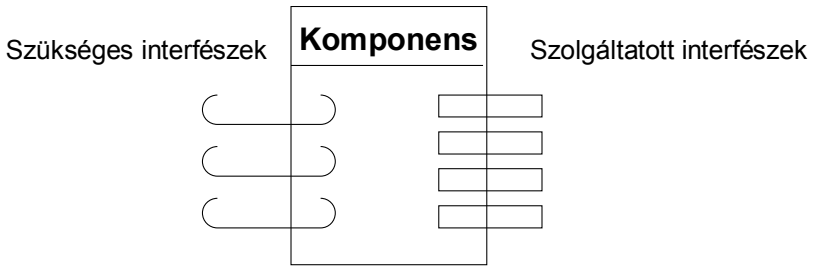
(Ebben az esetben elsősorban a szakterületi tudás újrafelhasználásáról van szó.)

A programgenerátorok típusai:
- A generátorok fajtái:
  - Alkalmazásgenerátorok - üzleti adatfeldolgozó rendszerek készítésére.
  - Szintaktikus elemzők - a programozási nyelvek értelmezésére.
  - CASE eszközökben lévő kódgenerátorok – egy szoftvertervből a tervezett rendszer implementációját állítják elő.
- A generátor alapú újrafelhasználás igen költség-hatékony, de viszonylag kevés szakterülethez léteznek ilyen rendszerek (üzleti adatfeldolgozás, eBusiness, stb.)
- Ezzel a módszerrel könnyebben állíthatók elő az alkalmazások, mint a komponens alapú módszerrel.

Komponens alapú fejlesztés:
- A komponens alapú szoftverfejlesztés (CBSE – Component Based Software Engineering) az újrafelhasználáson alapul.
- Kialakulásának oka az, hogy az objektumorientált fejlesztés nem támogatja az újrafelhasználást, mert:
  - Az egyedi objektumosztályok túl részletesek és specifikusak és csak a folyamat késői fázisában kapcsolódnak az alkalmazáshoz.
  - Nem alakult ki olyan piac, ahol az egyes szakterületek objektumosztályaihoz lehetne hozzájutni.
- A komponensek az objektumosztályoknál sokkal absztraktabbak és különálló szolgáltatásoknak tekinthetők.

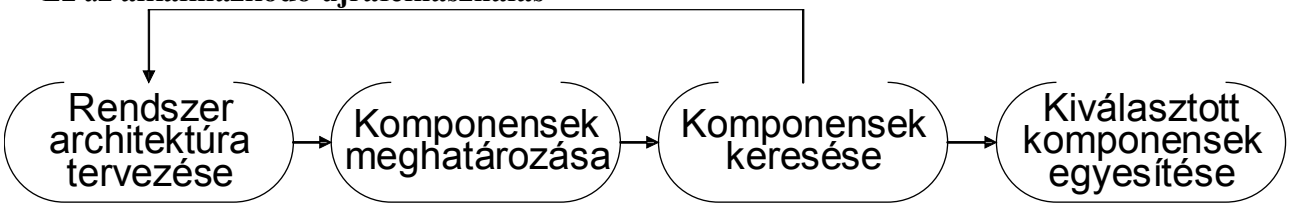
Komponensek:
- A komponensek szolgáltatásokat nyújtanak a rendszer számára, a végrehajtás helyétől és a megvalósítás nyelvétől függetlenül.
  - Egy komponens egy függetlenül végrehajtható program, amely egy vagy több végrehajtható objektumból áll.
  - A komponensek interfészeit publikálják és minden interakció ezeken az interfészeken keresztül folyik. A komponens forráskódja általában nem hozzáférhető, belső állapotai nem láthatóak.
- A komponensek mérete az egyszerű függvénytől a teljes alkalmazási rendszerig terjed.

A komponensek interfészei:
- **Szolgáltatott interfészek** – a komponens által szolgáltatott interfészek.
- **Szükséges interfészek** – azok az interfészek, amelyeket a komponenst használó rendszernek, vagy környezetének kell biztosítania.



A komponens alapú fejlesztési folyamat:

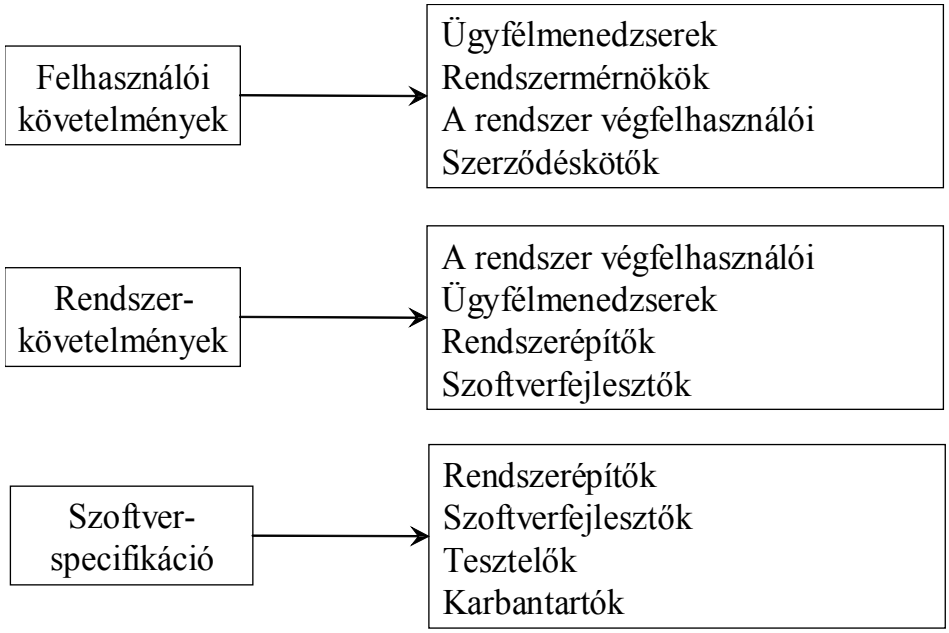
- A komponens alapú fejlesztés beilleszthető a szabályos szoftverfolyamatba, ha beépítjük abba az újrafelhasználással kapcsolatos tevékenységeket:
  - Komponensek specifikálása,
  - Komponensek megtalálása,
  - A tervek (esetleg a követelmények) módosítása a meglelt komponensek tulajdonságainak megfelelően.
- Ez az **alkalmazkodó újrafelhasználás**



7. A szoftverkövetelmények specifikálása, dokumentálása, formális specifikáció, interfészspecifikáció, viselkedésspecifikáció

- Felhasználói követelmények: A rendszer szolgáltatásainak közérthető leírása, diagramokkal, táblázatokkal, ábrákkal, a felhasználó számára.
- Rendszerkövetelmények: Strukturált dokumentum a rendszer szolgáltatásainak részletes leírásával. Ez a szerződés alapja.
- Szoftverterv specifikáció: A szoftver részletes leírása a fejlesztők számára

Specifikációk olvasói:



A felhasználói követelmények:
- A felhasználói követelményeket úgy kell megfogalmazni, hogy az informatikában járatlan felhasználó is megértse.
- Ezért itt nem célszerű modelleket alkalmazni, hanem természetes nyelven, űrlapokkal és diagramokkal kell a felhasználói követelményeket érthetővé tenni.
- A természetes nyelv alkalmazásának nehézségei:
  - Az egyértelműség és pontosság hiánya
  - A követelmények keveredése
  - A követelmények ötvöződése

Rendszerkövetelmények:
- A rendszerkövetelmények a felhasználói követelmények részletesebb leírását adják.
- A rendszertervezés alapjául szolgálnak, tartalmazhatják a rendszer modelljeit.
- Sokszor a szerződéshez csatolják, ezért a rendszer teljes és konzisztens meghatározását kell tartalmazniuk.
- A rendszerkövetelmények leírják, hogy a rendszernek mit kell elvégeznie, a tervek azt határozzák meg, hogy hogyan tegye.

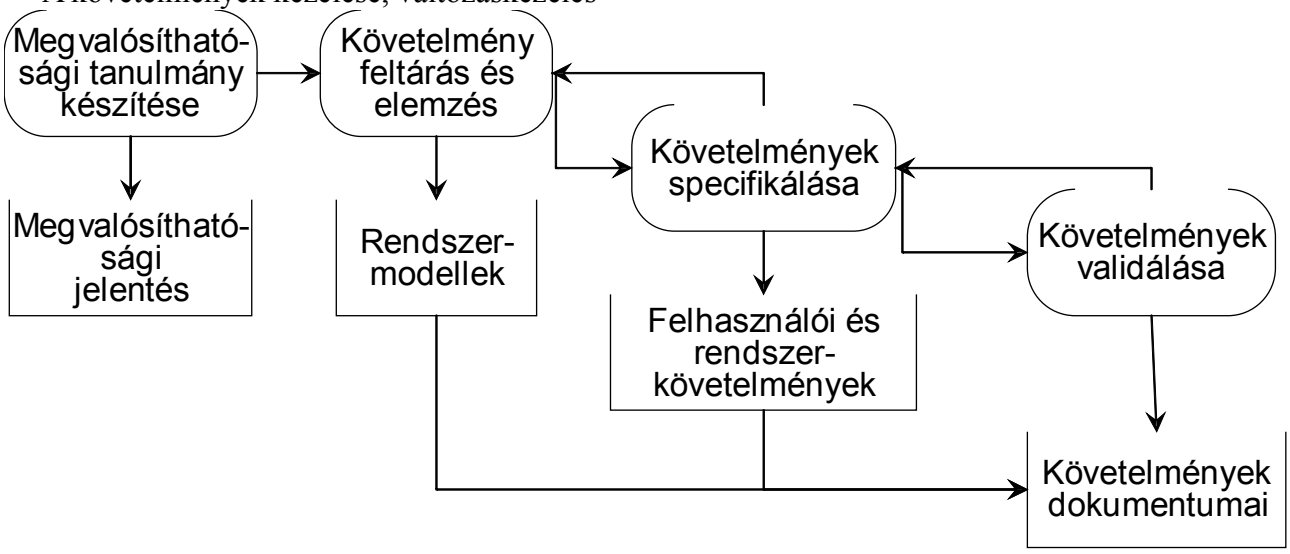
Szoftverterv specifikáció:
- A követelmények dokumentuma (System Requirements Specification – SRS) írja le, hogy mit várnak a tervezett rendszertől, vagyis mit kell megvalósítania a tervezőknek.
- A dokumentumnak nem a rendszer tervét kell tartalmaznia, hanem a követelmények definícióit

és specifikációját vagyis azt, hogy **mit kell tennie a rendszernek és nem azt, hogy hogyan**.

Követelmények precíz meghatározása:
- A követelményeket pontosan kell leírni, az olvasó számára egyértelműen.
- A pontatlan követelmény specifikáció félreértéseket eredményez.
- Például:
  - A felhasználó követelménye: „Megfelelő megjelenítőt kell biztosítani minden dokumentum típushoz”
  - A fejlesztő értelmezheti úgy, hogy csak egy szöveges megjelenítőre van szükség, így az összetett dokumentumokat nem lehet olvasni.

A követelmények teljessége, konzisztenciája:
- A követelményeket elvileg mindenre kiterjedően, és ellentmondásmentesen kell leírni.
- A teljesség azt jelenti, hogy a felhasználó által igényelt összes szolgáltatásnak szerepelnie kell.
- A leírás konzisztens, ha nincs konfliktus, vagy ellentmondás a rendszer szolgáltatásai között.
- A gyakorlatban ez a nagyméretű, komplex rendszereknél megoldhatatlan.

A követelménytervezés folyamatai:
- Megvalósíthatósági tanulmány
- A követelmények feltárása és elemzése
- A követelmények validálása
- A követelmények kezelése, változáskezelés



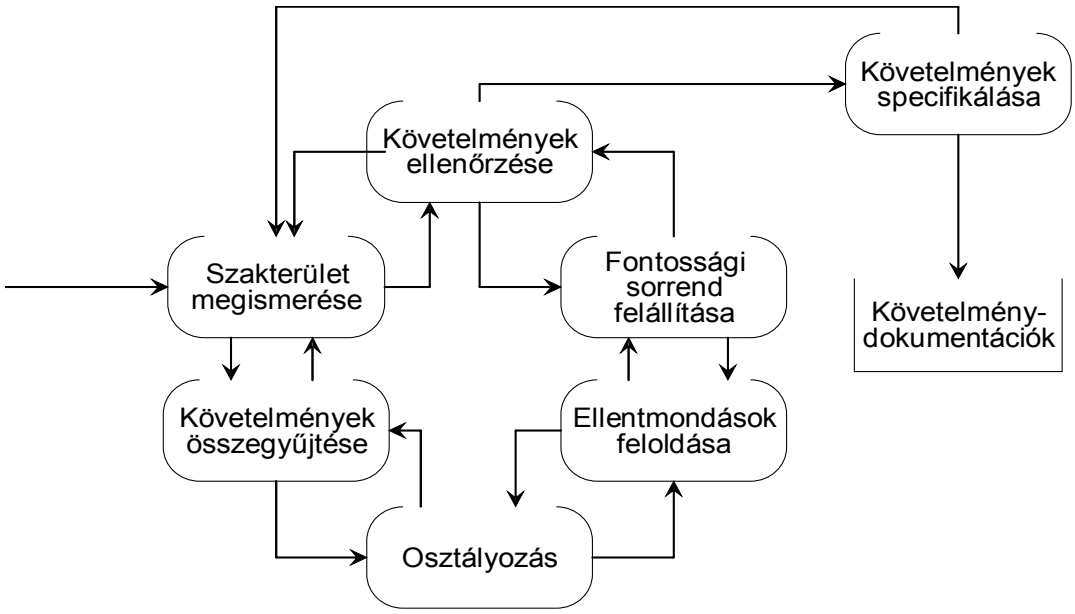
Megvalósíthatósági tanulmány:
- Feladata: megalapozni azt a döntést, hogy érdemes-e a tervezett rendszert megvalósítani.
- A tanulmány az alábbi kérdésekre ad választ:
  - Más, hasonló szervezeteknél milyen megoldásokat alkalmaztak hasonló feladatokra.
  - Mennyiben támogatja a rendszer a megrendelő általános célkitűzéseit.
  - Megvalósítható-e a rendszer a tervezett költségen belül, az adott technológiával, a kívánt határidőre.
  - Integrálható-e a rendszer más, már meglévő rendszerekkel.

Megvalósítósági tanulmány készítése:
- *Az alábbi információk felmérésén, összegyűjtésén alapul*
  - Milyen problémák vannak a jelenlegi folyamatokkal, és a tervezett rendszer hogyan segít ezeket feloldani.

  - Hogyan járul hozzá a rendszer a megrendelő üzleti céljaihoz.
  - Milyen folyamatokat kell támogatnia a rendszernek és melyeket nem.
  - Mi történne, ha a rendszert nem valósítanák meg.
  - Hogyan illeszkedik a rendszer a meglévő rendszerekhez.
- A tanulmány ajánlásokat tartalmaz a rendszer funkcióira és a megvalósítás módjára.

Követelmények feltárása és elemzése:
- Az informatikusok interjúkat készítenek a megrendelő kulcsembereivel (szakterületi képviselők, vezetők, és majdani felhasználók), hogy felderítsék, milyen szolgáltatásokat kell biztosítania a rendszernek.
- A követelmények feltárása bonyolult, mert:
  - A kulcsemberek gyakran nem tudják, hogy mit várhatnak és várnak egy számítógépes rendszertől.
  - A kulcsemberek a saját szakterületük fogalmait használják, a követelménytervezőknek ezeket kell megérteniük.
  - Az egyes szakterületeknek különböző elvárásai vannak.
  - Egyes kulcsfigurák a saját pozíciójuk erősítésére akarják felhasználni az új rendszert.
  - A környezet változása folyamatosan módosítja a követelményeket, a változásokat követni kell.

A feltárás és elemzés folyamata:



Formális specifikáció:
- A formális specifikáció egy matematikai jelölésrendszert alkalmaz, pontosan specifikált szótárral, szintaxissal és szemantikával.
- A specifikálás és a tervezés nagymértékben összefonódik egymással.
- Az architektúra-tervezés adhatja az alapot egy specifikáció struktúrájához.
- A szoftverspecifikáció folyamatának előrehaladásával az ügyfél befolyása csökken, a vállalkozó befolyása növekszik.

Formális specifikációs technikák:
- Algebrai megközelítés: A rendszert műveletek és azok kapcsolatai alapján írja le.
- Modell alapú megközelítés: A rendszert állapotmodellel specifikálja, amely halmazokból és

sorozatokból álló matematikai konstrukciókat tartalmaz, a műveleteket pedig aszerint definiálja, ahogy azok a rendszer állapotát módosítják.

Formális specifikáció alkalmazása:

- A formális specifikáció a szoftverfejlesztés kezdeti szakaszában kíván nagyobb erőfeszítéseket.
- A követelmények alaposabb és részletesebb elemzése azzal jár, hogy kevesebb lesz a hiba a követelményspecifikációban.
- A következetlenségek és a hiányosságok felfedhetők és kijavíthatók a formális modellekkel.
- Ezért a követelmények későn felfedezett hibáiból eredő többletmunka lesz kevesebb.

Interfész specifikáció:

- Egy új rendszernek legtöbbször más, már meglévő rendszerekkel kell együttműködnie. Az interfészeket a követelmények között kell specifikálni.
- Az interfészek típusai:
  - Procedurális interfészek – eljárások hívása egy másik rendszerből
  - Adatszerkezetek, amelyek az egyik rendszertől egy másikhoz kerülnek
  - Adatreprezentációk
- A formális jelölések egyértelművé teszik az interfészek definícióját.
- A nagy rendszereket alrendszerekre bontják, amelyek között jól definiált interfészeket kell specifikálni.
- Az alrendszerek közti interfészek specifikálása teszi lehetővé, hogy az alrendszerek fejlesztése egymástól függetlenül történjen.
- Az interfészek absztrakt adattípusokkal, vagy objektum osztályokkal definiálhatók.
- A formális specifikáció algebrai megközelítése különösen alkalmas az interfészek pontos specifikálására.
- A formális specifikációt nyílt interfészek, kommunikációs protokollok definiálására is alkalmazzák.

Viselkedésspecifikáció:

- Az algebrai specifikációs technikák nehézséget okozhatnak, amikor az objektumok műveletei nem függetlenek annak állapotától.
- A formális specifikációk gyakorlati alkalmazásában jobban elterjedtek a modell alapú specifikációs módszerek.
- A modell alapú specifikáció a rendszerspecifikációt a rendszer állapotmodelljeként fejezi ki. (Ilyen nyelvek többek között a VDM, a B és a Z)
- A **Z** a rendszereket halmazokkal és a halmazok közötti relációkkal modellezi. Kombinálja a formális és az informális leírásokat és grafikus ábrázolást alkalmaz.
  Különösen alkalmas interfészek és szoftver specifikálására. (ISO szabvány lesz)

8. A szoftverkövetelmények, a követelmények feltárásának, validálásának és kezelésének módszerei

lásd 5. tétel

9. A rendszermodellek típusai, környezeti és viselkedési modellek, adatmodell típusok

Rendszermodellezés:
- A rendszermodellezés segíti az elemzőket a rendszer funkcionalitásának megértésében.
- Egyes modellek alkalmazhatók a felhasználóval folytatott kommunikációban is.
- A különböző modellek eltérő nézőpontból ábrázolják a rendszert:
  - A környezeti modellek a rendszer környezetét és kapcsolatait mutatják be,
  - A viselkedési modellek a rendszer működését,
  - A szerkezeti modellek a rendszer felépítését, illetve az adatok szerkezetét ábrázolják.

Strukturált módszerek:
- A strukturált módszerek a módszer szerves részének tekintik a rendszermodellek készítését.
- Meghatározzák az elkészítendő modellek típusát és a modellre vonatkozó szabályokat és eljárásokat.
- Szabványokat adnak a készítendő dokumentumokra is.
- A módszert támogató CASE eszközök támogatják a modellkészítést és a dokumentálást.

A strukturált módszerek hátránya:
- Nem támogatják a nem-funkcionális rendszerkövetelmények megértését és modellezését.
- Nem tartalmaznak információt arra, hogy egy módszer alkalmazható-e az adott problémára.
- Gyakran túl sok dokumentációt eredményeznek. A követelmények lényege elvész a sok információ között.
- A rendszermodellek általában túl részletesek, a felhasználóik számára nehezen érthetők.

Modell:
- A modell absztrakt leírása egy olyan rendszernek, amelynek követelményeit előzőleg már összegyűjtötték, rendszerbe foglalták és elemezték.
- Az absztrakt modell jellemzője, hogy részleteket hagy el, egyszerűsít. Kiemeli a lényeget.
- Vagyis rendszermodell nem egy másik reprezentációja a rendszernek.

A rendszermodellek típusai:
- **Adatfeldolgozási modell:** Adatfolyam diagramok, az adatok feldolgozását mutatják a rendszeren belül.
- **Kompozíciós modell:** Egyed-kapcsolat diagramok. Bemutatják, hogyan épülnek fel az egyedek más egyedekből.
- **Architekturális modell:** Az alrendszereket mutatják be, amelyekből a rendszer felépül.
- **Osztálymodell:** Objektum osztály/öröklődési diagramok, az egyedek közös tulajdonságait ábrázolják.
- **Inger-válasz modell:** Állapotátmenet diagramok, a rendszer belső és külső eseményekre adott reakcióit írják le.

Környezeti modellek:
- A rendszer határainak ábrázolására szolgálnak (mi tartozik a rendszerhez és mi nem)
- A határok kijelölése gyakran nem technikai, hanem társadalmi, vagy szociális szempontoktól is függ.
- A rendszer és külső rendszerekkel való kapcsolatainak ábrázolása az architekturális modell feladata.
- A környezeti modell ábrázolási módja általában egyszerű blokkdiagram.

Folyamat modell:
- A teljes munkafolyamatot bemutatja.
- A folyamatmodell alapján lehet kijelölni, hogy a folyamat mely részeit kell támogatnia, vagy elvégeznie a számítógépes rendszernek.
- A folyamatok és a folyamatok közti információ áramlás bemutatására adatfolyam modellek használhatók.

Viselkedési modellek:
- A rendszer átfogó viselkedésének leírására szolgálnak.
- Típusai:
  - **Adatfolyam modellek:** Bemutatják, hogyan dolgozza fel a rendszer az adatokat.
  - **Állapotátmenet modellek**: Bemutatja, hogyan reagál a rendszer a különböző eseményekre
- A rendszer viselkedésének leírásához mindkét típusra szükség van.

Adatfolyam modellek:
- Modellezik az adatfeldolgozást a rendszerben.
- Azt mutatják be, hogyan áramlanak végig az adatok a feldolgozási lépések sorozatán és milyen átalakuláson mennek keresztül.
- Segítik az elemzőket abban, hogy megértsék, mi történik a rendszerben.
- Egyszerű jelölésrendszert alkalmaznak, ezért a megrendelő is könnyen megérti.
- Funkcionális szempontból modellezik a rendszert, és alkalmasak a rendszer külső adatkapcsolatainak ábrázolására is.

Állapotátmenet modellek:
- A rendszer viselkedését modellezik, a belső és külső eseményekre adott válaszokat írják le.
- Gyakran használják valósidejű rendszerek modellezésére.
- A rendszer állapotait csomópontként, az eseményeket nyilakkal jelöli. Egy esemény hatására a rendszer egyik állapotából egy másik állapotba kerül.
- Az állapotdiagramok az UML jelölésrendszer részét képezik.
- Feltételezi, hogy a rendszer egy adott időpontban a lehetséges állapotok egyikében van.

Az állapotátmenet strukturálása:
- Nagy rendszerekben az állapotok nagy száma miatt a modelleket strukturálni kell.
- A magasabb szintű modellben lévő szuperállapotok egy külön diagramon részletesen kifejthetők.
- Az alállapotok közti átmeneteket olyan ingerek válthatják ki, amelyeket a felsőbb szintű diagramon nem is jelölhetők, ezért az alállapotokat és az ingereket külön táblázatokban kell leírni.

Adatmodellek:
- A nagy rendszerek sokféle adatot tárolnak és dolgoznak fel, nagyméretű adatbázisokat alkalmaznak.
- Az adatbázisok sok esetben a rendszertől függetlenül léteznek, máskor a rendszerrel együtt kell létrehozni azokat.
- Az adatmodellek a rendszer által feldolgozott adatok logikai szerkezetének meghatározására szolgálnak. Az adatbázis tervezésben széles körben alkalmazzák őket.
- Leginkább elterjedt az **egyed-tulajdonság-kapcsolat** modellezése
- Az UML nem tartalmaz külön jelölésmódot az adatmodellezésre, az adatokat az objektumok és a köztük lévő kapcsolatok segítségével modellezi.
- Az egyedeket műveletekkel nem rendelkező, egyszerűsített objektumosztályoknak tekinthetjük,

így az UML osztálymodellje használható az adatok modellezésére is.
- Az adatmodelleket gyakran adatfolyam-modellekkel együtt használják

Viselkedési modell:

- Az objektumok viselkedése az általuk biztosított műveletek sorrendjének ábrázolásával történhet (szekvencia diagram).
- A szekvencia diagram voltaképpen egy forgatókönyv, amely a használati eseten alapul.
- A szekvencia diagramok mellett az UML-ben együttműködési diagramokat is használunk, ahol az objektumok által váltott üzenetek sorozatát ábrázoljuk.

10. Objektumorientált rendszermodellek, öröklődési modellek, az objektumok viselkedésének modellezése

A rendszermodellezés segíti az elemzőket a rendszer funkcionalitásának megértésében.
Egyes modellek alkalmazhatók a felhasználóval folytatott kommunikációban is.
A különböző modellek eltérő nézőpontból ábrázolják a rendszert:

- **A környezeti** modellek a rendszer környezetét és kapcsolatait mutatják be,
- **A viselkedési** modellek a rendszer működését,

A szerkezeti modellek a rendszer felépítését, illetve az adatok szerkezetét ábrázolják.
A modell absztrakt leírása egy olyan rendszernek, amelynek követelményeit előzőleg már összegyűjtötték, rendszerbe foglalták és elemezték.
Az absztrakt modell jellemzője, hogy részleteket hagy el, egyszerűsít. Kiemeli a lényeget.
Vagyis rendszermodell nem egy másik reprezentációja a rendszernek.

Adatfeldolgozási modell: Adatfolyam diagramok, az adatok feldolgozását mutatják a rendszeren belül.
Kompozíciós modell: Egyed-kapcsolat diagramok. Bemutatják, hogyan épülnek fel az egyedek más egyedekből.
Architekturális modell: Az alrendszereket mutatják be, amelyekből a rendszer felépül.
Osztálymodell: Objektum osztály/öröklődési diagramok, az egyedek közös tulajdonságait ábrázolják.

Inger-válasz modell: Állapotátmenet diagramok, a rendszer belső és külső eseményekre adott reakcióit írják le.

A rendszer átfogó viselkedésének leírására szolgálnak.

Viselkedési modellek típusai:

- **Adatfolyam modellek**:
  Bemutatják, hogyan dolgozza fel a rendszer az adatokat.
- **Állapotátmenet modellek**:
  Bemutatja, hogyan reagál a rendszer a különböző eseményekre

A rendszer viselkedésének leírásához mindkét típusra szükség van.

Objektum-modell

A rendszerkövetelményeket (főleg interaktív rendszerek esetében) gyakran objektum-modellel írják le. Az objektummodell objektumosztályokkal modellezi a rendszert. Az objektumosztály a közös tulajdonsággal rendelkező objektumok halmazának és az objektumok által nyújtott szolgáltatásoknak (műveleteknek) absztrakciója. Az objektumok az objektumosztály példányai, végrehajtható egyedek, az objektumosztály tulajdonságaival és szolgáltatásaival. A rendszerkövetelményeket (főleg interaktív rendszerek esetében) gyakran objektum-modellel írják le. Az objektummodell objektumosztályokkal modellezi a rendszert. Az objektumosztály a közös tulajdonsággal rendelkező objektumok halmazának és az objektumok által nyújtott szolgáltatásoknak (műveleteknek) absztrakciója. Az objektumok az objektumosztály példányai, végrehajtható egyedek, az objektumosztály tulajdonságaival és szolgáltatásaival.

Öröklődési modell

Az objektumosztályokat taxonómiába szervezi.
A taxonómia olyan osztályozási séma, amely megmutatja, egy osztály hogyan kapcsolódik más osztályokhoz, közös tulajdonságokon és szolgáltatásokon keresztül.
Az alacsonyabb szinten lévő osztályok:

- öröklik a magasabb szintű osztályoktól tulajdonságaikat és szolgáltatásaikat
- rendelkezhetnek speciális tulajdonságokkal és szolgáltatásokkal is.

Az oszály-hierarchia tervezés az egyik legnehezebb feladat, mivel az egyes ágakon a duplikálást kerülni kell.
Egyes objektumok más objektumokból épülnek fel, azok aggregátumaként.
Az aggregációs modell azt mutatja meg, hogy hogyan keletkezik több osztályból egy aggregált osztály.

Viselkedési modell

Az objektumok viselkedése az általuk biztosított műveletek sorrendjének ábrázolásával történhet (szekvencia diagram).
A szekvencia diagram voltaképpen egy forgatókönyv, amely a használati eseten alapul.
A szekvencia diagramok mellett az UML-ben együttműködési diagramokat is használunk, ahol az objektumok által váltott üzenetek sorozatát ábrázoljuk.

Case eszközrendszerek

Szoftver eszközök csoportja, amely a szoftverfolyamat egy, vagy több fázisát támogatja, mint az elemzést, implementációt, vagy tesztelést.
Az elemző és a tervező eszközök a modellezést segítik a követelménytervezés és a rendszertervezés során.
Ezek az eszközök vagy egy adott módszer támogatására készültek, vagy a diagramkészítést végzik.

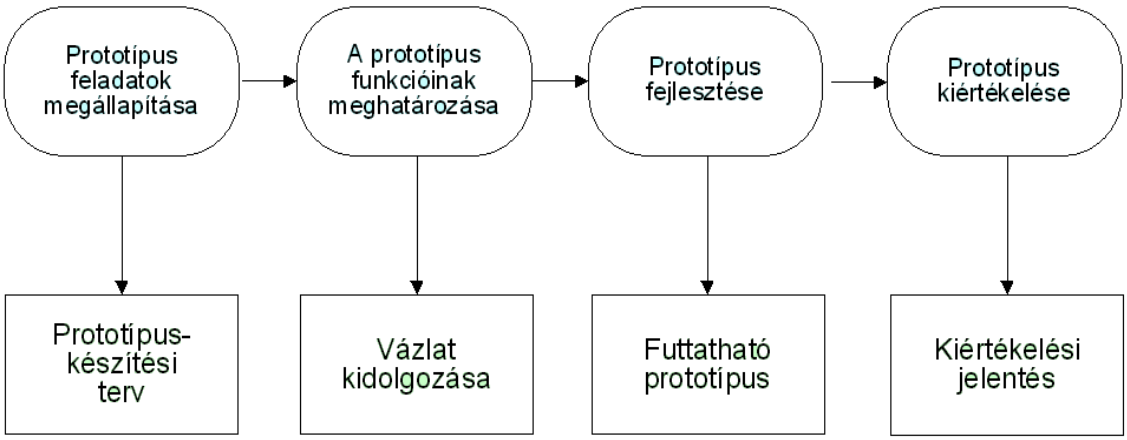
11. Szoftverprototípus készítése, a prototípusok fajtái, prototípuskészítés és adatbázis programozás

Rendszerprototípus készítése

A prototípuskészítés a követelménytervezés része, a követelmények feltárásának és validációjának eszköze.
A prototípus a szoftverrendszer kezdeti verziója, amely alkalmas a rendszer koncepciójának bemutatására és kipróbálására.

Korábban úgy tekintették, hogy a prototípus alacsonyabbrendű a kívánt rendszernél.
Ma a prototípus- és a normál rendszer közti határ fokozatosan elmosódik és sok rendszert az evolúciós modell alapján készítenek.

Prototípuskészítés folyamata:



A prototípusok alkalmazása

A felhasználó nem látja előre, hogyan fogja használni az új rendszert.
A prototípus elsődleges célja az, hogy segítse a felhasználókat a rendszerkövetelmények megértésében:

- A követelmények feltárása: a prototípussal a felhasználók megtapasztalhatják, hogyan támogatja a rendszer a munkájukat.
- A követelmények validálása: a prototípus felfedheti a hibákat és hiányosságokat a követelményekben.

A prototípus csökkenti a követelményekkel kapcsolatos kockázatokat.

A prototípuskészítés előnyei

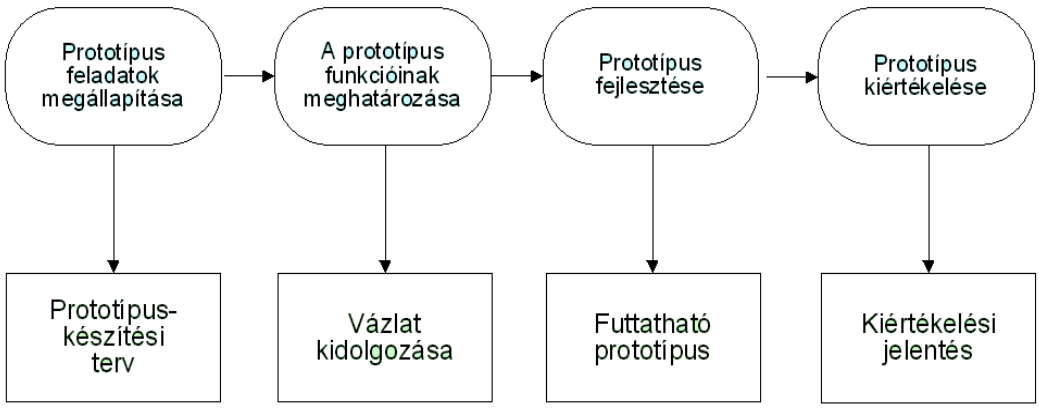
Felfedi a szoftver felhasználója és készítője közti félreértéseket.
Kiderülhet, hogy hiányzik valamely szolgáltatás, vagy ellentmondások vannak a szolgáltatások között.
A szoftverfolyamat elején már egy működő rendszer áll rendelkezésre.
A prototípus felhasználható a rendszerspecifikáció alapjaként.
Támogathatja a felhasználók képzését és a rendszertesztet.

A rendszer használhatóbb lesz.
A rendszer jobban illeszkedik a felhasználó igényeihez.
Javul a tervezés minősége.
Gyorsabban elkészül a rendszer.
A fejlesztéshez kevesebb erőforrásra van szükség.

- Evolúciós prototípus készítése:

Célja egy működő rendszer átadása a megrendelőnek.
A legfontosabb követelmények implementálásával egyszerű rendszer készül, amelyet újabb követelmények feltárásával fokozatosan egészítenek ki új funkciókkal.
Weblap fejlesztésben és e-business alkalmazásokban használják.

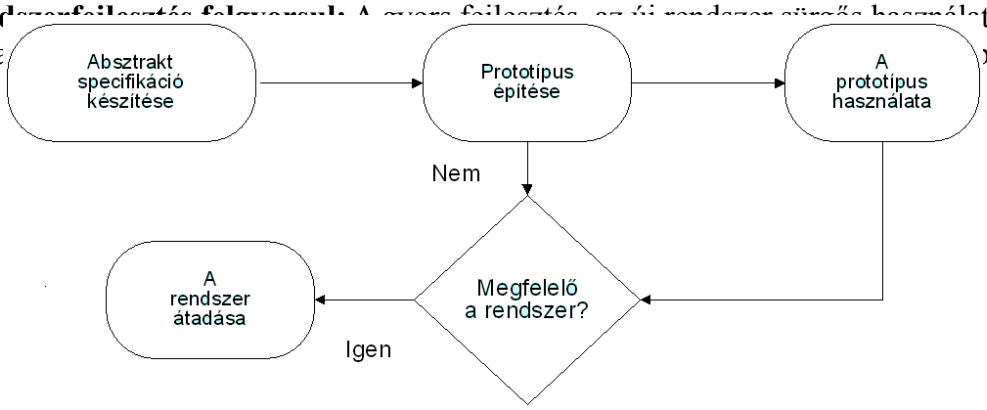
Olyan rendszereknél célszerű alkalmazni, ahol nem készíthető el előre a specifikáció. Ilyenek általában az intenzív felhasználói interfész használatot igénylő rendszerek.
Nincs részletes rendszerspecifikáció, sokszor a szabályos követelménydokumentum is hiányzik. A fejlesztéshez gyors, iterálható fejlesztő eszközökre és módszerekre van szükség. Mivel nem készül követelményspecifikáció, a validáció is csak a rendszer bemutatásával történhet.



Az evolúciós prototípuskészítés jellemzői

A specifikáció, a tervezés és az implementáció átlapolható.
A rendszer inkrementumok sorozataként fejlődik, és kerül a felhasználóhoz, vagyis a felhasználó kulcsfigurái minden inkrementum tervezésében és értékelésében részt vesznek.
Gyors fejlesztő eszközök és technikák alkalmazhatók (CASE eszközök és 4GL).
A felhasználói felületek GUI fejlesztő eszközökkel készíthetők.

A rend[illegible]tba vétele gyakra[illegible]antarthatóság



Növelhető a felhasználó elkötelezettsége: A felhasználók bevonása a rendszerfolyamatba azt eredményezi, hogy a rendszer nagyobb valószínűséggel felel meg az elvárásoknak, és a használatba vételkor a felhasználók már ismerik azt és tudják alkalmazni.

Az evolúciós prototípus hátrányai

Vezetési problémák: A jelenlegi vezetési módszerek a vízesés modellre alkalmazhatók. Az új technológiák alkalmazásához speciális ismeretekre, esetleg más munkatársakra van szükség.

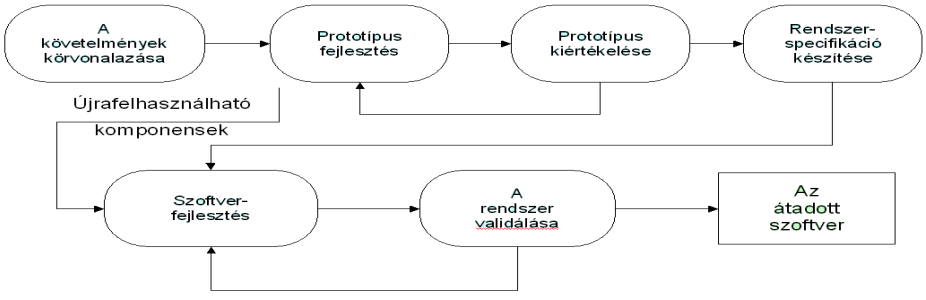
Karbantartási problémák: A folytonos változások a prototípus szerkezetének sérülését okozhatják, a dokumentáció hiánya és a speciális fejlesztő eszközök a karbantartást veszélyeztetik.

Szerződéskötési problémák: A fix áras szerződéshez előre ismerni kell a rendszer vázlatos követelményeit és tervét. A ráfordítás alapú szerződést pedig a megrendelő nem fogadja el.

Eldobható prototípus készítése: Célja a rendszerkövetelmények feltárása és validálása. A nem teljesen megértett követelmények megvalósítása és bemutatása segíti a feltárást. A követelményspecifikáció elkészülte után nem használható fel.

Célja a követelményspecifikációból fakadó kockázat csökkentése.
A prototípust egy kezdeti specifikáció alapján készítik, validálásra átadják a felhasználónak, majd eldobják.
Az eldobható prototípus nem tekinthető végleges rendszernek, mert:
Több rendszertulajdonság kimaradhat a prototípusból,
Nem készül specifikáció a hosszú távú karbantartásra.

A prototípus még nem a megfelelő struktúra szerint épül.



4. Gyors prototípuskészítési technikák

A gyors prototípuskészítéshez az alábbi technikák alkalmazhatók:
Fejlesztés dinamikus, magas szintű nyelven,
Adatbázisprogramozás,
Komponensek és alkalmazások összeépítése.
A gyakorlatban ezeket együttesen alkalmazzák.

A legtöbb prototípuskészítő eszköz tartalmazza a vizuális programozás támogatását, ahol grafikus szimbólumok reprezentálják a függvényeket, adatokat, feldolgozó szkrtipteket. Az eszköz a rendszer vizuális reprezentáció-jából generálja a végrehajtható programot.

Prototípuskészítő nyelvek

Szempontok az alkalmas nyelv kiválasztásához:
- Az alkalmazási terület jellege
  Természetes nyelvű feldolgozáshoz a Lisp, vagy a Prolog alkalmasabb.
- A felhasználói interakció jellege
  Az intenzív web alapú felhasználói interakció kidolgozására a Java, vagy Smalltalk több eszközt kínál.
- A támogatási környezet
  A nyelvvel együtt sokféle eszköz és komponens segíti a prototípuskészítést.

4.2 Adatbázis-programozás

Az evolúciós fejlesztés az adatbázison alapuló kis-, közepes üzleti alkalmazások területén általánosan alkalmazott technika.
A kereskedelmi adatbázis-kezelő rendszerek olyan 4GL fejlesztő eszközöket tartalmaznak, amelyek támogatják a lekérdezést/adatkezelést (SQL), táblázatkezelést, jelentés generálást, felhasználói felületek tervezését, stb.
Az adatfeldolgozási alkalmazásokban sok közös jegy van: adatbázis manipulációk (keresés, frissítés, rendezés, stb.), egyszerű műveletek, űrlapkezelés, stb. Egy 4GL-ben ezeket általánosítják. Gyakran integrálhatók CASE eszközökkel is. Ezek generálhatnak SQL-t, vagy alacsonyabb szintű kódot.

5. Felhasználói felületek prototípusai

A felhasználót be kell vonni a felhasználói felületek tervezésébe, a fejlesztő nem erőltetheti rá saját elképzeléseit.
A prototípusok készítése segít a felhasználók bevonásában.
A felhasználói interfész fejlesztése a rendszer-fejlesztési költségek növekvő hányadát adja.
Az interfész-generátorok segítenek abban, hogy a felhasználók gyorsan véleményt alkossanak a felületekről. A felületek specifikációjából jól strukturált programot generálnak.
A web alapú felhasználói interfészek készítésére jól ismert weblap tervező eszközök léteznek.

12. Architekturális tervezés, a rendszer strukturálása, rendszermodellek, vezérlési modellek

1. Az architekturális tervezés

Az architekturális tervezés az a tervezési folyamat, amelynek során kijelölik a rendszert alkotó alrendszereket és azt a keretrendszert, amely vezérli az alrendszereket és biztosítja közöttük a kommunikációt.
A folyamat végeredménye a szoftver architektúra, amely a tervezés alapjául szolgál.

1.1 A rendszerstruktúra meghatározása

A bonyolult rendszerek egymással lazán összefüggő részfeladatokból állnak, amelyek önállóan végrehajthatók, de egymással vezérlési és adatcsere kapcsolatban állnak.
- Példa: banki szolgáltató rendszer alrendszerei:

Központi feladatok: Ügyfélnyilvántartás, könyvelés, számlavezetés, betétkezelés, hitelkezelés, kártyakezelés, vezetői információs rendszer, stb.

Ügyfélkiszolgálással kapcsolatos feladatok:

- Ügyfél tranzakciók:
- Személyes kiszolgálás a bankfiókban, telefonos-, Internetes tranzakciók, kártyás vásárlások, ATM, pénzforgalom, hitelezés, értékpapír forgalom, stb. (egyéni- és vállalati ügyfelek számára)
- Bankközi tranzakciók: Átutalások, hitelek, fedezet-igazolás, értékpapír-műveletek, stb.

Az architekturális tervezés

A rendszertervezés folyamatának kezdeti lépcsőfoka. Feladata:

Összekötni a specifikáció és a tervezés folyamatát.
Kialakítani a rendszer alapvető struktúráját és azt a keretrendszert, amely a rendszert egységbe foglalja és működését irányítja.

Gyakran egyes specifikációs tevékenységgel párhuzamosan végezhető.
Magába foglalja a fő rendszerkomponensek és azok vezérlésének valamint kommunikációjának meghatározását.

A jól megtervezett architektúra előnyei

A tervezői megbeszélések alapját képezi
A tervezés kulcsszereplői számára érthetővé teszi a rendszer vázát.

Támogatja a kritikus kérdések korai elemzését
Az architektúra terv alapján megítélhető, hogy a rendszer eleget fog-e tenni olyan kritikus követelményeknek, mint a teljesítmény, megbízhatóság, karbantarthatóság, skálázhatóság.
Megalapozza az újrafelhasználhatóságot
Az alrendszerekre bontás és azok fő tulajdonságainak meghatározása lehetőséget ad újrafelhasználható komponensek kifejlesztésére (vagy felhasználására), termékcsaládok kidolgozására, amelyben az azonos feladatokat újrafelhasználható komponensek oldják meg.

Alrendszer

Az alrendszer olyan – szolgáltatásaik alapján – egységként kezelhető komponensek rendszere, amely önállóan oldja meg feladatát.
Modulokból, vagy más alrendszerekből áll, szabványos interfészen keresztül veheti igénybe más alrendszerek szolgáltatásait.

Modul

Olyan rendszer-komponens, amely szolgáltatás(oka)t nyújt más moduloknak és igénybe veszi mások szolgáltatásait, de nem tekinthető független alrendszernek.
Más, egyszerűbb modulokból (komponensekből) áll.

A rendszer architektúrája kihat a nem-funkcionális rendszerkövetelmények kielégítésére, így meghatározza a:

Teljesítményt: Egy rendszer teljesítménye jobb lesz, ha nagyméretű modulokból áll, mert kevesebb kommunikáció zajlik a modulok között.
A védelmet: A jobb védelem érdekében rétegezett szerkezetet célszerű alkalmazni, a kritikus eszközöket a legbelső rétegben elhelyezve.
A biztonságot: A biztonsággal kapcsolatos műveletek egy, vagy néhány alrendszerben legyenek.
A rendelkezésre állást: Redundáns komponensek alkalmazásával növelhető.
A karbantarthatóságot: Sok önálló, könnyen változtatható komponensből kell felépülnie.

2. A rendszer strukturálása

A rendszer együttműködő alrendszerekké való felbontása.
Az architektúra terv általában egyszerű blokkdiagram formájában ábrázolja a rendszer (mindenki által megérthető) struktúráját.
Részletesebb modellek is alkalmazhatók, amelyek megmutatják:

hogyan osztják meg egymás közt az alrendszerek az adatokat,
Hogyan kommunikálnak egymással.

2.1 Tárolási modell

Az alrendszerek két módon cserélhetnek információt egymással:

- A megosztott adatok egy központi adatbázisban vannak, amelyet minden alrendszer elérhet. Ez a tárolási modell (repository).
- Minden alrendszernek van saját adatbázisa, és az alrendszerek üzenetek formájában cserélnek adatokat.

A nagy adatmennyiséggel dolgozó rendszerek legtöbbször osztott adatbázis köré szervezett alrendszerekkel dolgoznak.Ilyenek például a nagy, vállalatirányítási rendszerek, CASE és CAD rendszerek, stb.

2.2 Kliens-szerver architektúra

Olyan osztott rendszermodell, amely bemutatja hogyan oszlanak meg az adatok és a feldolgozások a komponensek között. Elemei:

Szerverek: Adatkezelő szerverek, nyomtatószerverek kommunikációs szerverek, stb.
Kliensek: Többnyire önálló alrendszerek, amelyek hozzáférnek a szerverek szolgáltatásaihoz. Egyszerre sok példányban futnak.
- Vékony kliens (böngésző, szkriptekkel)
- Vastag kliens (komplett kis alrendszer, helyi adatokkal)

Hálózat: A klienseknek biztosít hozzáférést a szerverek szolgáltatásaihoz.

2.3 Absztrakt gép modell (réteges modell)

Az alrendszerek közti interfészek modellezésére használják.
Rétegekbe (absztrakt gépekbe) szervezi a rendszert, amelyek mindegyike adott szolgáltatásokat végez.
Támogatja az egyes alrendszerek inkrementális fejlesztését. Az egyes rétegek egyszerűen kicserélhetők, csak az interfészek szabályait kell betartani, de annak változtatásához is csak a két szomszédos réteget kell módosítani.
Előnye, hogy mivel a hardvert, operációs rendszert a belső rétegekbe zárja, könnyen adaptálható különböző platformokra. (protokoll modellek:ISO-OSI)
Hátránya: strukturálása bonyolult, egy külső réteg csak a közbensőkön keresztül férhet hozzá a legbelsőkhöz.

3. Vezérlési modellek

A strukturális rendszermodellek nem tartalmaznak vezérlési információkat, az alrendszerekre való felbontást ábrázolják.
A vezérlési modellek az alrendszerek közötti vezérlési folyamatokat modellezik.
- **Központosított vezérlés:** Egy alrendszer végzi a teljes rendszer vezérlését, indítja, leállítja, stb. a többi alrendszert.
- **Esemény alapú vezérlés:** Minden alrendszer reagálhat az őt érintő külső vagy más alrendszer által generált eseményekre.

3.1 Központosított vezérlés

Hívás-visszatérés modell: Fa-struktúrájú modell, ahol a csúcson van a vezérlő alrendszer. A vezérlés hívások sorozatán keresztül jut el a modulokhoz. Szekvenciális rendszerekhez alkalmazható (pl. listafeldolgozás, listázás, jelentésgenerálás).

Kezelő modell: Konkurens rendszerek modellezésére alkalmas. Egy központi rendszerkomponens koordinálja, indítja, állítja le a rendszerfolyamatokat (komponenseket, vagy alrend-szereket), amelyek párhuzamosan is végrehajthatók. Alkalmazható szekvenciális rendszerekben is, ahol a vezérlő modul állapotváltozók értéke alapján hívja meg az egyes alrendszereket.

3.2 Eseményvezérelt rendszerek

A környezet által generált események irányítják a rendszert. Az esemény nemcsak bináris jel, hanem érték változása is lehet. Az esemény időzítése az eseményt feldolgozó alrendszer hatályán kívül esik.

Broadcast modell: Az eseményről mindegyik alrendszer értesül, és az reagál rá, amelyiknek ez a feladata.

Megszakításvezérelt modell: Valós idejű rendszerek modellje, ahol egy megszakítás-kezelő észleli az eseményt és elindítja az esemény feldolgozásáért felelős alrendszert.

Eseményvezérelt rendszer lehet pl. egy táblázatkezelő is, ahol egy cella értékének megváltozása más cellá-kat is megváltoztat, vagy más alrendszert aktivizál.

13. Objektumorientált tervezés, UML diagramok, az objektuminterfész specifikáció

Az objektumorientált tervezés lényege

Az objektumok a valódi világ vagy egy rendszer elemeinek absztrakciói, amelyek karbantartják saját állapotukat.
Az objektumok függetlenek, de együttműködnek egymással, „elrejtik" állapotukat és jellemzőiket.
A rendszer funkcionalitása az objektumok szolgáltatásaiban fejezhető ki.
Nem használnak megosztott adatterületeket , üzenetek útján kommunikálnak egymással.
Az objektumok szétoszthatók, szekvenciálisan, vagy párhuzamosan végrehajthatók.

Az objektumorientált tervezés előnyei

Könnyebb karbantartani: Az objektumok önálló egységekként értelmezhetők.
Az objektumok megfelelő újrafelhasználható komponensek.
Az egyes rendszerekben egyértelmű leképezés van a való világ elemei és a rendszer objektumai között.

Az objektumorientált fejlesztés

Az objektumorientált fejlesztés az alábbi – összefüggő, de különálló - fázisokból áll:

Objektumorientált elemzés (OOA): Az alkalmazás objektumorientált modelljének kidolgozása.

Objektumorientált tervezés (OOD): A követelményeknek megfelelő szoftverrendszer objektumorientált modelljének kidolgozása.

Objektumorientált programozás (OOP): A szoftverterv objektumorientált programnyelven történő megvalósítása.

Az egyes fázisok között nincs explicit határvonal, egy következő lépés az előző finomításával jár.
Egy objektum egy olyan entitás, amely jól körülhatárolható, állapota van és meghatározott műveletek tartoznak hozzá. Az állapotait az attribútumainak értékkészlete határozza meg. A műveletekkel szolgáltatásokat nyújt más objektumoknak.
Az objektum az objektumosztály egy példánya.
Az objektumosztály definíciója az objektumok templétjének tekinthető. Tartalmazza az attribútumok és szolgáltatások deklarációit, amelyek az osztályhoz tartozó objektumokhoz köthetők.

Az objektum:

Azonosítható: Az objektumok egymástól megkülönböztethetők, függetlenül az állapotuktól.
Tulajdonságok, attribútumok tartoznak hozzá: Ezek lehetnek kötött, formális paraméterek is.
Állapot tartozik hozzá: Az attribútumok konkrét értékei az objektum mindenkori állapotát határozzák meg.
Műveletek tartoznak hozzá: Ezek lehetnek leképezések, tevékenységek, események.
Korlátozott láthatósággal rendelkezik: Van látható része (export és import műveletek), van

láthatatlan része (az ábrázolás és a szolgáltatások megvalósításának részletei).

Az UML modellezési nyelv

UML – Unified Modeling Language az Objectum Oriented Analisys & Design (OOA&D) eszköze.

Az UML: egy rendszer grafikus ábrázolására alkalmas eszközök (diagramok) gyűjteménye. Ezek többségét már korábban is alkalmazták, amelyek az UML-ben egységes jelölésrendszerbe kerültek, így kialakult egy grafikus nyelv, amely – legalábbis a jelölésrendszert ismerők számára – jól áttekinthető specifikációk, modellek, tervek és dokumentációk készítésére ad lehetőséget, amely többé-kevésbé automatikus transzformációkkal objektum orientált nyelvű programokká alakítható.

Az objektumorientált tervezés során, az utóbbi 20 év során kidolgozott jelölések egységesítésével létrejött modellezési nyelv.
Jelölésrendszerével az objektumorientált analízis és tervezés során készíthető modellek ábrázolását támogatja.
Az objektumorientált tervezésben de-facto szabvánnyá vált.

UML diagramok

Statikus szempont szerint:
- **Osztálydiagram** (Class diagram): a rendszer objektumelvű szerkezetének leírása
- **Objektumdiagram** (Object diagram): Az osztálydiagram egy példányát mutatja be

Dinamikus szempont szerint:
- **Állapotdiagram** (Statechart): A rendszer állapotait, állapotátmeneteit mutatja be
- **Szekvenciadiagram** (Sequence diagram): Az objektumok közötti üzenetváltások időbeni menetét ábrázolja
- **Aktivációs diagram** (Activity diagram): Az objektumok és a tevékenységek egymásra gyakorolt hatását mutatja be
- **Együttműködési diagram** (Collaboration diagram): Az objektumok együttműködését ábrázolja

Implementációs szempont szerint:
- **Komponensdiagram** (Component diagram): A komponensekből felépülő rendszert mutatja be
- **Alrendszerdiagram** (Subsystem diagram): Az alrendszerek kapcsolatát ábrázolja

Környezeti szempont szerint:
- **Konfigurációs diagram** (Deployment diagram): A rendszer környezetét, a hw/sw konfigurációt szemlélteti

Felhasználói szempont szerint:
- **Használati eset diagram** (Use case diagram): A rendszer és a felhasználók kapcsolatát mutatja be

Objektumosztály

Jellemzői:

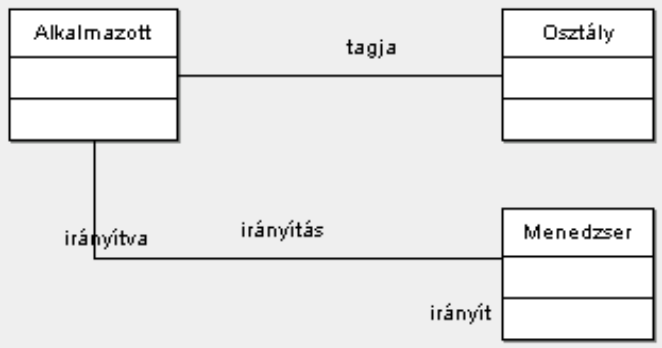
Hasonló tulajdonságú objektumok halmaza: Szerkezeti és viselkedésbeli jellemzők hasonlósága.
Az osztálynak van neve: A nevet az osztályba tartozó összes objektum örökli.
Lehetnek attribútumai, paraméterei: Hozzáférési módok: public, private, protected.
Tartoznak hozzá szolgáltatások, műveletek (export): Az osztály minden objektumára, vagy az osztály egészére vonatkozó műveletek.
Tartozhat hozzá import felület: Az általa igényelt szolgáltatások definíciói.
Rendelkezhet megvalósítási résszel:A megvalósítás leírása.
Van látható és láthatatlan része
Lehet absztrakt vagy konkrét osztály
Lehet paraméteres (sablon) osztály

Asszociáció az UML jelöléseivel

Az objektumok és objektumosztályok kapcsolatban vannak más objektumokkal és objektumosztályokkal.
Az UML-ben az asszociációkat az objektumosztályok közötti vonallal és a kapcsolatot leíró megjegyzésekkel modellezik.

Az asszociációk általánosak, de jelezhetik, hogy:
- egy objektum egy attribútuma egy vele kapcsolatban álló objektum, vagy
- egy műveletének implementációja egy vele kapcsolatban lévő objektumtól függ.

Egy asszociációs modell



8.2.5 Az objektumok interfész specifikációja

Az objektumok közti interfészt úgy kell specifikál-ni, hogy az objektum és más komponensek tervezése párhuzamosan folyhasson.
Az interfészek reprezentációjának rejtettnek kell lennie, hogy változtatható legyen. Műveleteket kell biztosítani az adatokhoz való hozzáférésre és módosításra.
Egy objektumnak többféle interfésze is lehet, amelyek az objektum szolgáltatásainak különféle nézőpontból való megközelítését teszik lehetővé.
Az UML osztálydiagramokat használ az interfészek specifikációjára, de a Java szintén használható

14. Felhasználói felületek tervezése, alapelvek, megjelenítés, felhasználói támogatás

1. A felhasználói kezelőfelület

A felhasználó a kezelőfelületen keresztül kerül kapcsolatba a rendszerrel, ennek alapján alkot véleményt, csak ezután ismeri meg a rendszer funkcionalitását.
A rosszul tervezett kezelőfelület gyakran katasztrofális hibákhoz vezet.
A szegényes, vagy következetlen felhasználói kezelőfelület sok rendszer bukásához vezetett.
Nagy fejlesztő szervezetekben szakértőket alkalmaznak (grafikus, pszichológus, szakterületi szakértő), de kis/közepes cégeknél a kezelőfelület megtervezése is a szoftver tervező feladata.

Grafikus felületek

A korai rendszerek csak alfanumerikus terminálokat alkalmazhattak, a kezelőfelület karakteres, vagy űrlap jellegű volt. Már ekkor kialakultak a kezelőfelületekkel szembeni alapkövetelmények:

> Legyen strukturált, következetes, áttekinthető,
> Biztosítson segítő szolgáltatásokat,
> A hibákat egyértelműen jelezze.

Ma csaknem minden rendszer nagyfelbontású, színes, grafikus felületet támogat. Az interakcióra nemcsak a klaviatúra, hanem egér, vagy más kijelölő eszköz is rendelkezésre áll.

A grafikus kezelőfelület előnyei

Könnyebben megtanulható és használható, akár számítógépes ismeretek nélkül is.
A felhasználó több képernyőt használhat az interakcióra, gyorsan válthat különböző alkalmazások között, az információ látható maradhat az éppen nem aktív ablakban is.
A felhasználó a teljes képernyő bármely részét elérheti, ez gyors interakciót tesz lehetővé.

2. A felhasználói kezelőfelületek tervezésének alapelvei

A kezelőfelület tervezésekor figyelembe kell venni a felhasználók igényeit, gyakorlatát és képességeit.
Az emberek fizikai és mentális képességei korlátozottak (rövid távú memória), a felhasználói felület tervezésekor ezt figyelembe kell venni.
A grafikus felhasználói felületek tervezésének alapelvei minden felhasználói interakció tervezésének alapjául szolgálhatnak.

A felhasználói jártasság figyelembevétele: A felületnek olyan kifejezéseket és fogalmakat kell használnia, amelyeket az átlagos felhasználó ismer.

A felület konzisztenciája: A menüknek és parancsoknak ugyanazzal a formátummal kell rendelkezniük, hasonló műveleteket hasonló módon kell megvalósítani.

Minimális meglepetés: A felhasználóban kialakul egy modell a rendszer működéséről. A hasonló tevékenységeknek hasonló hatást kell kiváltaniuk, különben a rendszer kellemetlen meglepetéseket

okoz felhasználó számára.

Visszaállíthatóság: Minden helyzetben számítani kel arra, hogy a felhasználó hibázhat, ezért gondoskodni kell arról, hogy a hibát kijavíthassa:

- Visszavonási lehetőség (undo), esetleg többszintű,
- Veszélyes (pl. törlés) tevékenységek megerősítése,
- „Puha törlés”

A felhasználó támogatása: A felületnek könnyen elérhető segítő, vagy súgó rendszerrel kell rendelkeznie. A súgót strukturálni kell, nem szabad túl sok információt közölni. Előnyös a helyzetfüggő súgó alkalmazása.

A felhasználó sokfélesége: Az alkalmi felhasználók több támogatást, a gyakorlott felhasználók egyszerűbb, gyorsabb működést várnak.

4. Az információ megjelenítése

A rendszer megjeleníti a felhasználó számára közlendő információkat.
Ez az információ megjelenhet közvetlenül szöveges formában, vagy más módon (pl. grafikusan, akár hang kíséretében).
A jól tervezett rendszerekben maga az információ és az azt megjelenítő szoftver különválik.
A Model-View-Controller (MVC) általánosan alkalmazott architektúra az adatok többféle megjelenítését.
Az információ lehet:

Statikus információ: Értéket kap a munkafázis (session) kezdetén és ez a session ideje alatt nem változik meg, lehet numerikus, vagy szöveges

Dinamikus információ: Megváltozik a munkafázis alatt és a megváltozott értéket a felhasználó számára meg kell jeleníteni, lehet numerikus, vagy szöveges
A megjelenítés stílusában meg kell különböztetni őket.

A figyelmeztetés megjelenítésekor a grafika kiemeli a fontos szöveget, az információ jellegére ikonnal is utalhatunk.

A szöveg és grafika mellett hang is használható a figyelem felkeltésére, amennyiben feltételezhető, hogy a felhasználók nagy része rendelkezik hangkártyával.

5. Felhasználói támogatás

A felhasználó támogatása kiterjed a rendszer minden megjelenési formájára: súgó, hibaüzenetek, kézikönyvek, stb.
A felhasználó tájékoztatását be kell építeni a felhasználói felületekbe, hogy minden helyzetben kérhessen támogatást, vagy kapjon információt, ha hibát vétett.
Célszerű a súgó és az üzenő rendszert összeépíteni, hogy minden üzenetről magyarázatot kérhessen a felhasználó.
A hibaüzenetek tervezése különösen fontos: a kezdő felhasználó ezekkel találkozik a leggyakrabban. A rossz, vagy számára érthetetlen hibaüzenetek miatt elutasíthatja a rendszert.
Az üzeneteknek udvariasnak, előrevivőnek és következetesnek kell lennie.
A felhasználó háttere, gyakorlata a hibaüzenetek tervezésének meghatározó tényezője.

15. Osztott rendszerek architektúrái, többprocesszoros architektúrák, kliens-szerver architektúrák

Osztott rendszerek

A hálózatok terjedésével lassan minden rendszer (még a beágyazott rendszerek is) más rendszerekkel kapcsolatban működik. (Eltűnnek az egyedülálló rendszerek?)

Az osztott rendszerek jellemzői:

Erőforrásmegosztás (oda kell fordulni, ahol létezik a kívánt szolgáltatás : WebServices!)
Nyíltság (többféle hw/sw szállító termékeit tartalmazzák)
Konkurencia (az egyes gépekben párhuzamos folyamatok mennek végbe, amelyek időnként kommunikálnak és szinkronizálják egymást)
Skálázhatóság
Hibatűrés
Átlátszóság (a felhasználó nem látja, hogy osztott rendszerrel van kapcsolatban), **de** esetenként szükség van arra, hogy a felhasználó tisztában legyen vele, honnan vesz igénybe erőforrásokat, szolgáltatásokat (pl. web-w-es alkalmazások nagy része)

Az osztott rendszerek hátrányai

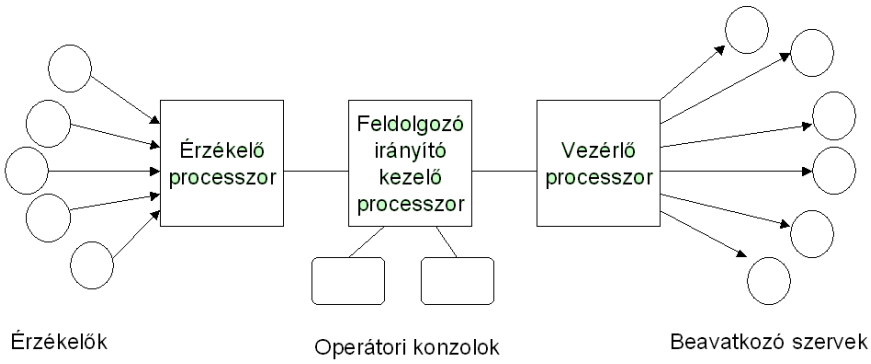
Bonyolultság nehezebb a rendszer és tulajdonságinak (pl. teljesítmény) tervezése, karbantartási nehézségek, stb.
Kezelhetőség a különböző hardver és operációs rendszer operálása nagy nehézségeket okozhat.
Biztonság az osztott rendszer biztonságát sokszor szintén elosztva kell megoldani. (segítenek a modern rendszerek SSO (single-sign-on) megoldásai)

Osztott rendszerek tervezési kérdései

Erőforrások azonosítása: Névkonvenciókra van szükség, hogy megtalálhatók és hivatkozhatók legyenek az erőforrások (pl. interneten URL)
Kommunikáció: Az internet, TCP/IP sok mindenre megfelelő, de néha speciális kommunikációs protokollokra van szükség (valósidejű közvetlen kapcsolatok)
A szolgáltatás minősége: Sok tényezőtől függ (hw, op.rendszer/ek, architektúra: erőforrások elosztása, hálózat, a rendszer rugalmassága)
Szoftverarchitektúra: A funkciók elosztása a rendszer logikai komponensei között, ezek eloszlása a hardver erőforrások között (pl. adatbázis szerver önmagában többprocesszoros rendszeren futhat.)
A logikai komponensek között köztes szolgáltatásra (middleware) van szükség.

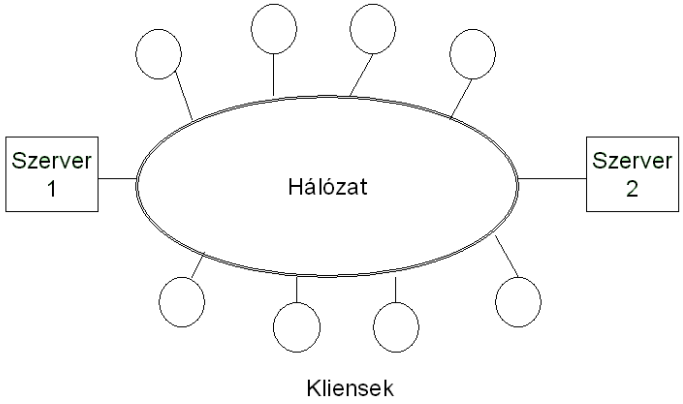
Többprocesszoros architektúrák

A különböző folyamatok külön processzorokon futnak. Példa: ipari folyamatirányítás



Kliens-szerver architektúrák

A klienseknek tudniuk kell a szolgáltatásokról (szerverekről), de egymásról nem kell tudniuk.
A kliens-szerver architektúrának az alkalmazás logikai szerkezetét kell tükröznie (és nem a fizikai gépeket)



16. Verifikáció és validáció, a verifikáció tervezése, verifikációs és validációs módszerek

A szoftver verifikációja és validációja

Verifikáció: Annak ellenőrzése, hogy valóban a megfelelő terméket készítjük el, vagyis, hogy a szoftver megfelel a specifikációnak.

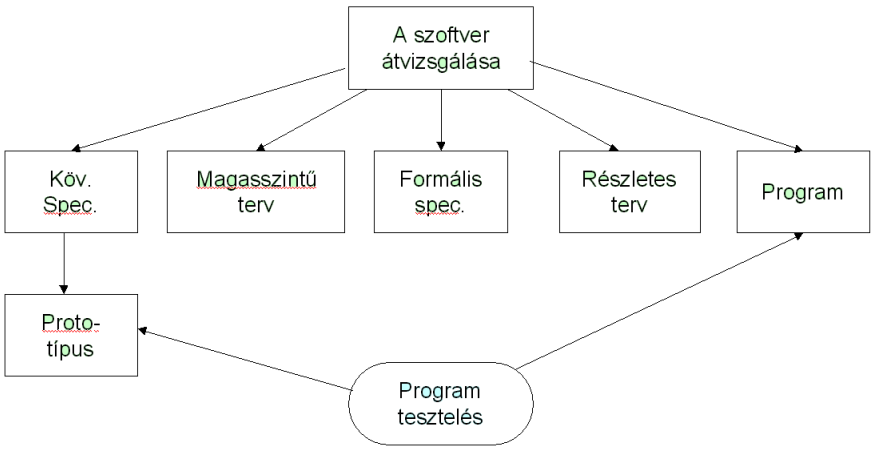
Validáció: Annak bizonyítása, hogy a terméket jól készítjük el, vagyis hogy a szoftver valóban a megrendelő elvárásainak megfelelően működik (esetleg a specifikációval ellentétesen).

A szoftvernek azt kell megvalósítania, amit a felhasználó valóban elvár tőle.
A verifikáció és validáció (V&V) folyamata a szoftver teljes életciklusára kiterjed, a szoftver folyamat minden fázisában szerepet kap.
Alapvetően két célja van:

Felfedni a rendszerben rejlő hibákat
Meggyőződni arról, hogy a rendszer egy-egy konkrét működési szituációban használhatóan működik.

A V&V folyamatban kétféle technika alkalmazható:

Szoftver-átvizsgálás (inspekció): A rendszer reprezentációjának elemzése (Köv. Spec., Tervek, grafikus ábrázolások, forráskód). A forráskód elemzése automatizálható.
Szoftvertesztelés: A szoftver implementációjának tesztadatokkal való futtatása és a viselkedés megfigyelése (dinamikus verifikáció)



1.2 Programtesztelés

Még ma is a legelterjedtebb validációs technika (bár a szoftverfolyamat végén helyezkedik el). A hiba meglétét kell felfedeznie nem a hiba hiányát. Az a sikeres teszt, amely legalább egy hibát felfedez. Az egyetlen módszer a nem-funkcionális követelmények validálására. A statikus (szemlék) verifikációval együtt kell alkalmazni.
Hiányosságok tesztelése

Feladata a rendszer hibáinak és hiányosságainak felfedése.

Fajtái:

Komponens tesztek: Fekete doboz, ekvivalencia-osztályok, struktúrateszt, útvonal-teszt
Integrációs tesztek: „fentről lefelé/lentről felfelé”, interfészteszt, stressz-tesztek
Objektumorientált tesztelés

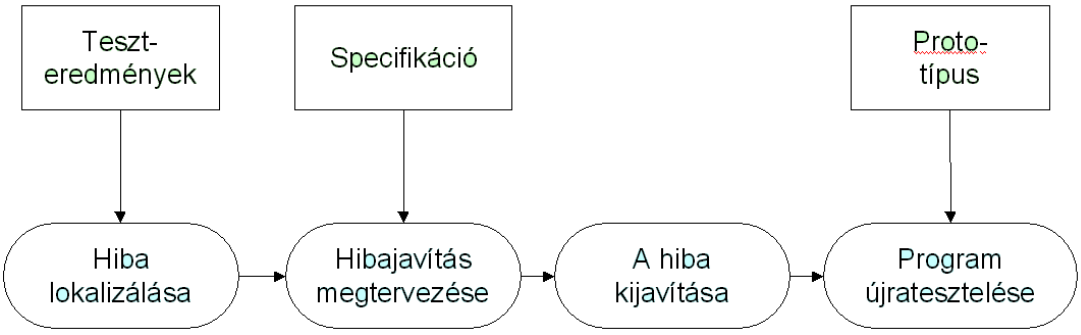
Például egy interaktív rendszer esetén tesztelni kell:
A menükön elérhető összes funkciót,
Egyazon menüponton elérhető valamennyi rendszerfunkciót,
A felhasználói inputok által használt összes függvényt, helyes és helytelen input adatokkal egyaránt.

Statisztikai tesztelés: A rendszer teljesítményének és megbízhatóságának tesztelése, valós helyzetekben (valós felhasználói inputtal és gyakorisággal).

A V&V célja: megbizonyosodni arról, hogy a szoftver rendszer megfelel a céljának. (Vagyis nem az, hogy hibamentes!)

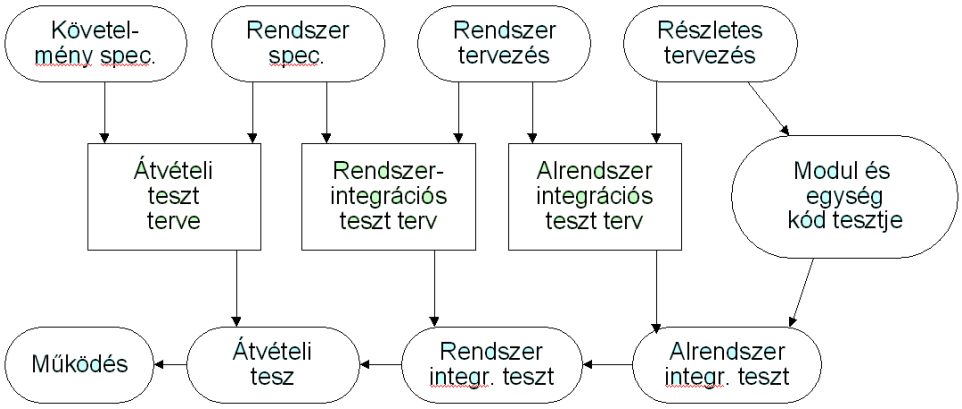
A hiányosságok tesztelése és a belövés különböző folyamatok:
A verifikáció és validáció feladata a hibák, hiányosságok létezésének felfedezése.
A belövés ezen hibák helyének lokalizálása és kijavítása.

A belövés a program viselkedésére vonatkozó feltételezések felállításával kezdődik, majd ezen feltételezések vizsgálatával próbálja megtalálni a hibákat.
Belövés folyamata:



2. A verifikáció és validáció tervezése
Alapos tervezésre van szükség, hogy a legtöbb eredményt kapjuk az egyébként igen költséges tesztelésből és felülvizsgálatból.
A V&V tervezését a fejlesztési folyamat elején meg kell kezdeni.
A tervnek meg kell határoznia az arányokat a statikus verifikáció és a tesztelés között.
A teszt-tervezésre a nagyobb cégeknél a általános szabványokat, szabályokat dolgoznak ki. Ennek alapján kell megtervezni és végrehajtani a termék konkrét tesztelését.

A tesztelés és a fejlesztés modellje



A szoftver átvizsgálása

A szoftver átvizsgálás célja a hiányosságok felderítése, a költséges tesztelés helyett a hibák kb. 60 %-a felfedhető az átvizsgálás során.
A fejlesztési folyamat kezdetétől alkalmazható, a dokumentumok (követelmények, tervek) átvizsgálásával.
Egy átvizsgálás során több hiányosság felfedezhető, amíg egy teszt többnyire egy hibát fed fel.
A tapasztalt vizsgálók (inspektorok) már ismerik és könnyen megtalálják a típushibákat

Átvizsgálás és tesztelés

Az inspekció és a tesztelés nem helyettesítik egymást, de a korai fázistól rendszeresen végzett átvizsgálás sok költséges tesztet előzhet meg.
Mindkettőt alkalmazni kell a V&V folyamatban.
Az inspekció alkalmas eszköz arra, hogy ellenőrizze, megfelel-e a program a specifikációnak.
A nem-funkcionális rendszerkövetelmények vizsgálatára azonban a felülvizsgálat nem használható.

Programátvizsgálás

A dokumentumok átvizsgálásának formalizált eszköze: Tapasztalt szakemberek nézik át a dokumentumokat és a kódot, ellenőrző lista alapján.
Célja a hiányosságok felderítése a forráskódban (logikai hibák, kezdőérték nélküli változók, szabványoknak való meg nem felelés, stb.)
Az átvizsgálást követően a programozó módosítja a programot, az új verziót nem kell feltétlenül újravizsgálni.

Statikus elemző

A statikus elemzők a forráskódot vizsgáló szoftver eszközök.
Nem futtatják a programot, hanem elemzik a program szövegét.

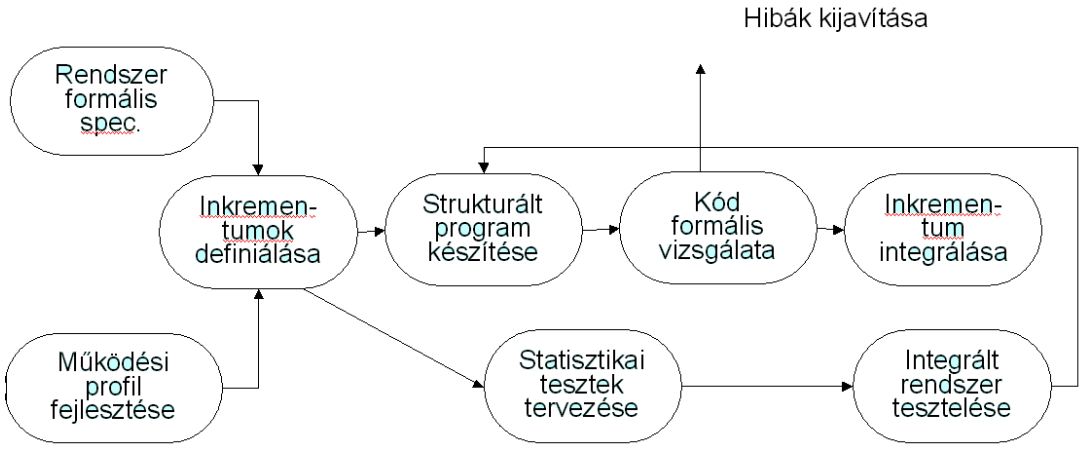
2.4 Cleanroom szoftverfejlesztés

A szoftverhibák elkerülését, nem pedig megtalálá-sát és kijavítását célzó szigorú átvizsgálási folyamat. (A név a félvezető gyártásból származik)
A rendszer komponenseinek tesztelését helyettesíti átvizsgálásokkal, megfelelnek-e a specifikációnak.
Inkrementális fejlesztési módszer, először a kritikus inkrementumokat szállítja le.

A Cleanroom jellemzői:
Formális specifikáció (állapotátmenet modell, strukturált programozás, csak néhány vezérlési és

adatabsztrakciós kontrukció használható)
Inkrementális fejlesztés
Statikus verifikáció (szigorú átvizsgálások)
A rendszer statisztikai tesztelése

Cleanroom folyamata:



17. Szoftvertesztelés, a hiányosságok tesztelése, tesztelés és belövés, integrációs tesztelés

A szoftvertesztelés

A tesztelési folyamat alapelemei:

Komponens tesztelés: Az egyedi programkomponensek tesztelése. Általában a komponens fejlesztőjének feladata (a kritikus rendszerek kivételével)

Integrációs tesztelés: Komponensek csoportjának tesztelése, amelyek egy rendszert vagy alrendszert alkotnak. Független tesztelő csoport feladata. A tesztek a rendszer specifikációja alapján készülnek

A funkcióorientált rendszereknél:

A rendszer alapvető program-egységei (függvények – modulok) jól elkülöníthetők,
Ezek külön tesztelhetők.

Az objektumorientált rendszerek esetén:

Az ilyen megkülönböztetés nem lehetséges, az objektumok lehetnek egyszerű (pl. lista), vagy komplex entitások (pl. egy alrendszer objektumai),

Olykor nincs egyértelmű hierarchia az objektumok között, ezért az integrációs tesztek (fentről lefelé, vagy lentről felfelé) nem alkalmazhatók.

A hiányosságok tesztelése

A cél: feltárni a rejtett hibákat a rendszerben.

A sikeres hiányosság teszt a rendszer helytelen működését eredményezi (ellentétben a validációs teszttel, amely a rendszer helyes működését ellenőrzi).

A hibák jelenlétét és nem azok hiányát mutatja ki.

Tesztadatok és tesztesetek:

A **tesztesetek** a teszthez szükséges inputok és a vért outputok specifikációi,
A **tesztadatok** a rendszer tesztelésére kidolgozott input adatok.

A tesztelés súlyponti kérdései

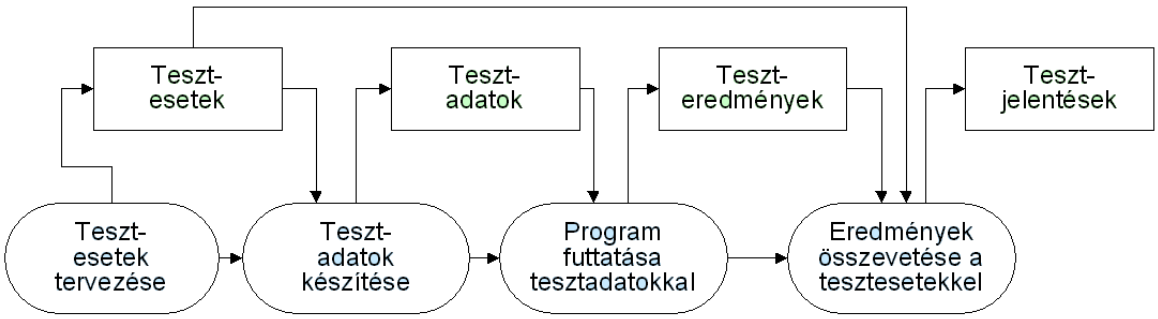
Csak teljes körű tesztelés bizonyíthatná, hogy a rendszer hibamentes, de a teljes tesztelés lehetetlen.

A teszteknek a rendszer a képességeit kell vizsgálniuk, nem a komponenseket.

A fejlesztés során a rendszer régi képességeinek tesztelése fontosabb, mint az újonnan hozzáadott képességeké.

A tipikus helyzetek tesztelése fontosabb, mint a határesetek tesztelése.

A hiánosságok tesztelésének folyamata

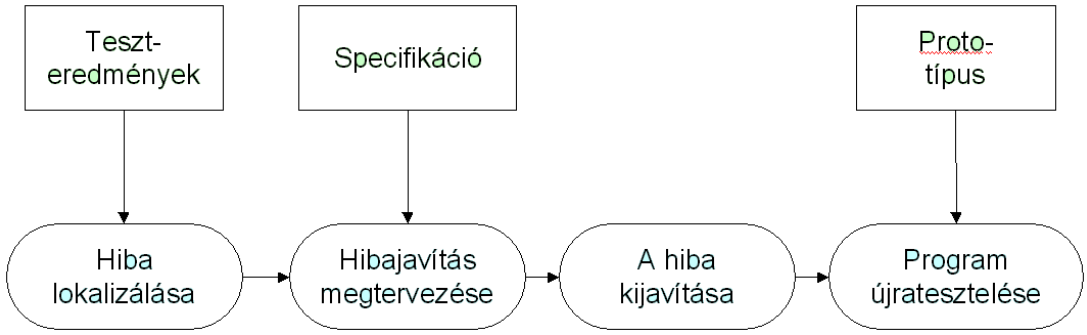


Tesztelés és belövés

A hiányosságok tesztelése és a belövés különböző folyamatok:
A verifikáció és validáció feladata a hibák, hiányosságok létezésének felfedezése.
A belövés ezen hibák helyének lokalizálása és kijavítása.

A belövés a program viselkedésére vonatkozó feltételezések felállításával kezdődik, majd ezen feltételezések vizsgálatával próbálja megtalálni a hibákat.

Belövés folyamata:



2. Integrációs tesztelés

Teljes rendszerek vagy alrendszerek tesztelése, amelyek előzőleg már tesztelt komponensekből állnak.
A komponensek együttműködéséből származó hibák feltárására szolgál.
Az integrációs teszt fekete doboz tesztelés, a tesztek a specifikációból származnak.
Komplex rendszerben az észlelt hibás eredményből nehéz a hiba helyére következtetni.
Az inkrementális integrációs tesztelés némileg segít.

2.1 Az integrációs tesztelés stratégiái

Fentről lefelé tesztelés: A rendszer magas szintű komponenseit még a tervezés és az implementáció alatt integrálják. A még el-nem készült komponenseket azonos interfésszel készült „csonkok” helyettesítik ahol szükséges. Ezeket fokozatosan kicserélik a kész elemekkel. (Evolúciós fejlesztésnél alkalmazható)

Lentről felfelé tesztelés: A hierarchia alsó szintjein lévő modulok integrálásával és

tesztelésével kezdik, ahol a magasabb szinteket tesztgenerátorok helyettesítik. (Inkrementális és újrafelhasználás alapú fejlesztésnél)

A gyakorlatban a kettő kombinációját alkalmazzák.

18. A szoftver minőség fogalma, minőségbiztosítási szabványok, a minőség tervezése, szoftverkarbantartás

A szoftver minőségének kezelése

A minőségbiztosítás célja, hogy garantálja a szoftvertermék megkívánt minőségét.

A minőségbiztosításhoz tartozik:
a minőség szabványainak és a betartásukhoz szükséges folyamatoknak meghatározása,
annak dokumentálása, hogy a termék a szabványoknak megfelelően készült.

Az megfelelő minőség eléréséhez „minőségi kultúrát" kell kialakítani, amelyért mindenki felelősséggel tartozik.

A szoftver minősége

A minőség általában azt jelenti, hogy a termék megfelel specifikációjának.

A szoftver esetében ezt nehéz értelmezni, mert:
Eltérő a megrendelő és a fejlesztő minőségi elvárása:
A megrendelő azt várja, hogy a szoftver legyen gazdaságos, megbízható, stb.
A fejlesztő minőségi követelményei: karbantarthatóság, újrafelhasználhatóság.
Egyes minőségi kritériumokat nem lehet egyértelműen definiálni (pl. karbantarthatóság, hordozhatóság)
A szoftver specifikációját nehéz teljessé tenni, tehát a specifikációnak való megfelelés nem garantálja, hogy a felhasználó elégedett lesz a termékkel.

A minőségkezelés tevékenységei

Minőségbiztosítás: Szabványok és szervezeti eljárások alkalmazása.
Minőségtervezés: Egy konkrét projekthez alkalmas eljárások és szabványok kiválasztása és adaptálása.
Minőségellenőrzés: Annak biztosítása és ellenőrzése, hogy a fejlesztő csapat alkalmazza a minőségi szabványokat és eljárásokat.

A minőségkezelés lehetőleg legyen független a projektvezetéstől.

Az ISO 9000 szabvány

A minőségkezelés nemzetközi szabvány-rendszere
(ISO – International Standard Organisation)
Sokféle szervezetre alkalmazható, a termeléstől a szolgáltatásokig.
Az ISO 9001 alkalmazható a tervezéssel, fejlesz-téssel, karbantartással foglalkozó szervezetekre.
A megújított ISO 9000:2000 szabvány már foglalkozik a felhasználói elégedettség mérésével is.
Az ISO 9001 egy általános minőségkezelési folyamat, amelyet adaptálni kell a konkrét szervezetre.

Az ISO 9000 tanúsítvány

A vonatkozó minőségi szabványokat és eljárá-sokat a szervezet minőségi kézikönyvében kell lefektetni.

Egy független, külső bizottság tanúsítja, hogy a szervezet minőségi kézikönyve és gyakorlata megfelel az ISO 9000 szabványnak.
A tanúsítást évente felülvizsgálják és megújítják, vagy megvonják a szervezettől.
A megrendelők (pl. közbeszerzésben) mind gyakrabban írják elő feltételként az ISO 9000 tanúsítványt.

A minőségbiztosítás és a szabványok

A szabványok adják a keretet a hatékony minőségkezeléshez.
Lehetnek: nemzetközi-, nemzeti-, szervezeti- és projektszabványok.

A **termékszabványok** olyan tulajdonságokat írnak elő, amelyek a termék minden elemére nézve kötelezőek:
- Dokumentációs szabványok (pl. dokumentumok szerkezete)
- Kódolási szabványok (programozási stílus, programnyelv használat)

A **folyamatszabványok** a szoftverfejlesztés alatt követendő folyamatokat határozzák meg (pl. a specifikáció, tervezés, stb. folyamata, módszerei, dokumentumai).

Minőségtervezés

A minőségi tervet a folyamat korai szakaszában kell elkészíteni.
A minőségi terv meghatározza a termék minőségi jellemzőit, kijelöli a mérés módját és az alkalmazandó folyamatokat.
Meg kell határozni, hogy milyen szervezeti szabványokat kell alkalmazni.
Ha szükséges, új szabványokat dolgoznak ki.

A minőségi terv szerkezete

A minőségi terv tartalma: A termék bemutatása, Terméktervek, A folyamatok leírása, Minőségi célok, Kockázatok és kockázatkezelés

A minőségi tervnek rövidnek, tömörnek kell lennie (különben nem olvassák el!).