

## 8. Programfejlesztési modellek

### Nagy rendszerek fejlesztési fázisai, kapcsolataik

#### Fejlesztési fázisok

##### 1. A probléma megoldásának előzménye

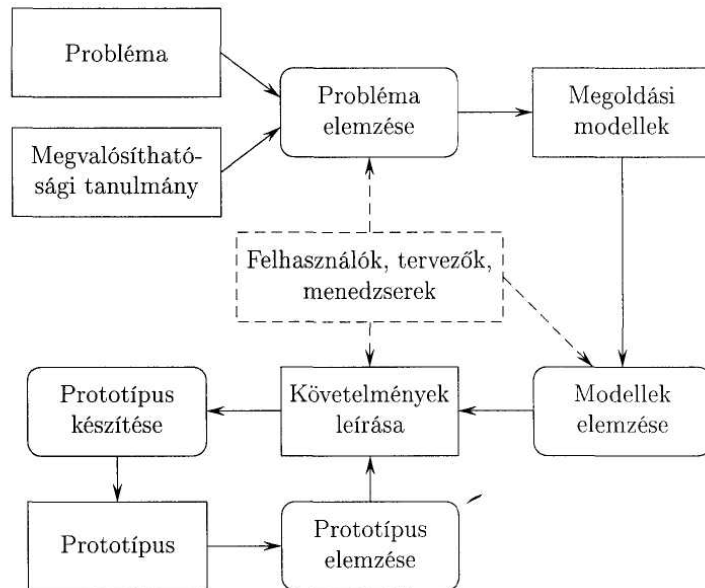
Egy probléma megoldása előtt meg kell vizsgálni a megvalósíthatóságát, és annak mikéntjét.

Eredmény: *Megvalósíthatósági tanulmány*, mely a következőkre válaszol:

- Erőforrások (hardver, szoftver, szakember)
- Költségek
- Határidő
- Üzemeltetés

##### 2. Követelmények leírása

Rendszerint iteratív módon állítjuk elő, és a prototípust használjuk a finomításra (ábra 1).



1. ábra. Követelményleírás elkészítésének folyamata

## Követelmények leírásának tartalma:

- Probléma
- Korlátozó tényezők (hardver, szoftver, stb.)
- Elfogadható megoldás

## Követelmények leírásának fajtái:

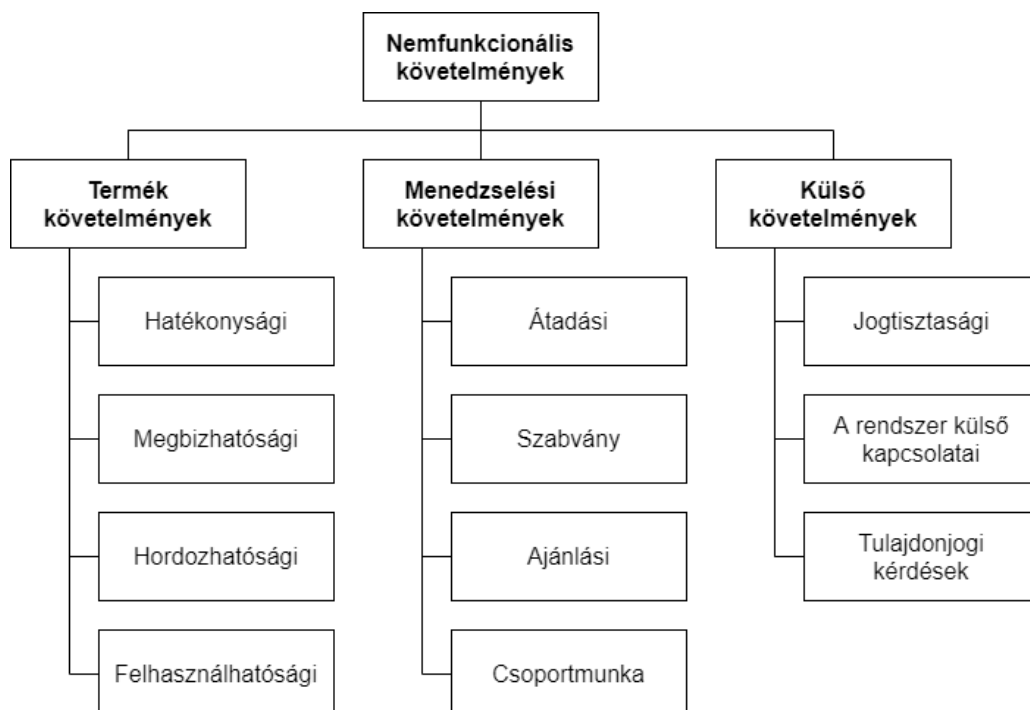
- ***Funkcionális követelmények***

A rendszer szolgáltatásainak, leképzéseinek leírása:

- Elindítás formája
- Bemenő adatok (és azok megadásának formája)
- Igénybevétel előfeltétele, korlátozások
- Szolgáltatás kezdeményezésére a válasz, eredmények
- Válasz megjelenési formája
- Bemenő adatok és válasz közti reláció

- ***Nem funkcionális követelmények***

A nem funkcionális követelményeket rendszerint három osztályba soroljuk: a termék követelményei, menedzselési követelmények, külső követelmények. Az osztályokat tovább lehet bontani (ábra 2).



2. ábra. Nem funkcionális követelmények osztályozása

### 3. Követelmények elemzése és prototípus

A következőket kell megvizsgálni:

- Önmagában jó-e a követelmények leírása?
  - Konzisztens (nincs ellentmondás)
  - Komplett (teljes)
- Validáció vizsgálat  
(Megfelel-e a felhasználó által elképzelt problémának?)
- Megvalósíthatósági vizsgálat  
(A követelményeknek megfelelő megoldás megvalósítható-e?)
- Tesztelhetőségi vizsgálat  
(A követelmények úgy vannak-e megfogalmazva, hogy azok tesztelhetők?)
- Nyíltság kritériumainak vizsgálata  
(A követelmények nem mondanak-e ellent a módosíthatóság, a továbbfejleszthetőség követelményének?)

A követelmények elemzésének egyik eszköze a prototípus-készítés. A prototípus magas szintű programozási környezetben létrehozott, a külső viselkedés szempontjából helyes megoldása a problémának.

### 4. Programspecifikáció

A programspecifikáció a következő kérdésekre kell, hogy válaszoljon a követelmények leírása alapján:

- Mik a bemenő adatok? (Forma, jelentés, megjelenés.)
- Mik az eredmények? (Forma, jelentés, megjelenés.)
- Mi a reláció a bemenő adatok és az eredmény adatok között?

### 5. Tervezés

A tervezés során a következő kérdésekre adjuk meg a választ:

(a) Statikus modell

- Rendszer szerkezete
- Progamegységek, azok feladata és kapcsolata

(b) Dinamikus modell

- Hogyan oldja meg a rendszer a problémát?
- Milyen egységek működnek együtt?
- Milyen üzenetek játszódnak le?
- Rendszer és egységek állapotai
- Események (melyek hatására állapotváltás történik)

(c) Funkcionális modell

- Milyen adatáramlások révén valósulnak meg a szolgáltatások?
- Milyen leképezések játszanak szerepet az adatáramlásokban?
- Mik az ajánlások az implementáció számára?
  - Implementációs stratégiára vonatkozó ajánlás.
  - Programozási nyelvre vonatkozó előírás, ajánlás.
  - Tesztelési stratégiára vonatkozó ajánlás.

A gyakorlatban két tervezési módszer terjedt el: *procedurális* és a *objektumelvű*

- *Procedurális*: megvalósítandó funkciókból, műveletekből indulunk ki, és ezek alapján bontjuk fel a rendszert kisebb összetevőkre, modulokra
- *Objektumelvű*: a rendszer funkciói helyett az adatokat állítjuk a tervezés középpontjába. A rendszer által használt adatok felelnek meg majd bizonyos értelemben az objektumoknak.

## 6. Implementáció

Fontos szempontok:

- Reprezentáció (Adatok ábrázolása)
- Események leképezések megvalósítása
- Algoritmusok és optimalizálások

Az implementáció egyik alapvető kérdése az implementációs stílus. A jó programozási stílus néhány fontos eleme:

- absztrakció különböző szintjeinek alkalmazása
- öröklődési technika használata, absztrakciós szintek hierarchikus rendszere
- absztrakciós szintekre bontás osztályon belül (deklaráció + megvalósítás)
- korlátolt láthatóság
- információ elrejtés (information hiding)
- információ beburkolás (encapsulation)

## 7. Verifikáció, validáció

A rendszer eleget tesz-e a vele szemben támasztott elvárásoknak?

Verifikáció: a specifikációszerinti helyesség igazolása

Validáció: Minőségi előírások teljesítése (robusztusság hatékonyság, erőforrásigény)

Ennek folyamata: tesztelés, melynek szakaszai:

- Egységteszt
- Rendszerteszt

A tesztelésnek két módja lehet:

- fekete doboz - Csak a maguknak a hibáknak a felderítése. Nem veszi figyelembe a program belső működését.
- fehér doboz - Hibák helyének felderítése. Figyelembe veszi a belső működést.

## 8. Rendszerkövetés és karbantartás (maintenance)

Karbantartás: Üzemebe helyezés után szükségessé váló szoftver jellegű munkák [pl.: rejtett hibák kijavítása, adaptációs munkák (új hardver-, szoftverkörnyezet), továbbfejlesztési munkák]

Rendszerkövetés: a felhasználókkal való kapcsolattartás menedzsment jellegű, dokumentációs feladatai [pl.: konfigurációk nyilvántartása, verziók menedzselése, dokumentáció menedzselése]

## 9. Dokumentáció

Egy nagy méretű program önmagában nem tekinthető szoftverterméknek dokumentáció nélkül. Egy jó dokumentáció a következőképp épül fel.

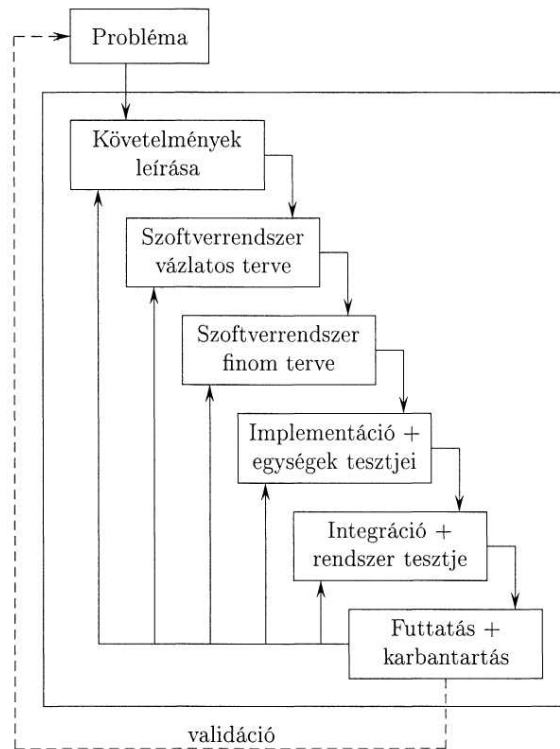
- Felhasználói leírás
  - Feladateleírás
  - Futtató környezet
  - Fejlesztések, verziók
  - Installálás
  - Használat
  - Készítők
- Fejlesztői leírás
  - Modulok (és azok szerkezete)
  - Osztályok (és azok kapcsolata)
  - Rendszer dinamikus viselkedése
  - Osztályok implementálása (adatszerkezetek, sablon osztályok)
  - Tesztelés

## Fejlesztési fázisok kapcsolatai

A fejlesztési fázisok leírására többféle modellt használhatunk

### 1. Vízesés modell

Az egyes fázisok egymást követik, a módosítások a futtatási eredmények ismeretében történnek. Egy bizonyos fázisban elvégzett módosítás az összes rákövetkező fázist is érinti.



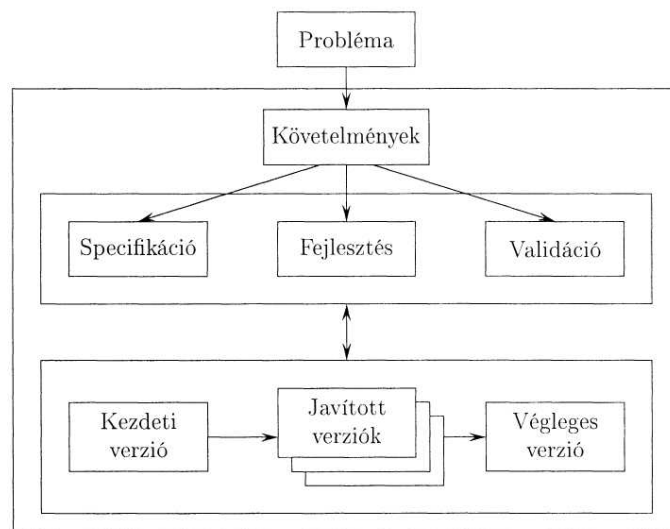
3. ábra. Vízesés modell

Hátrányai:

- Új szolgáltatás minden fázison módosítást igényel
- Validáció az egész életciklus megismétlését követelheti meg

## 2. Evolúciós modell

A megoldást közelítő verzióinak, prototípusainak sorozatát állítjuk egymás után elő, és így haladunk lépésenként egészen a végleges megoldásig. Ennek során egy verzió elkészítésekor a specifikáció, a fejlesztés és a validáció párhuzamosan történik.



4. ábra. Evolúciós modell

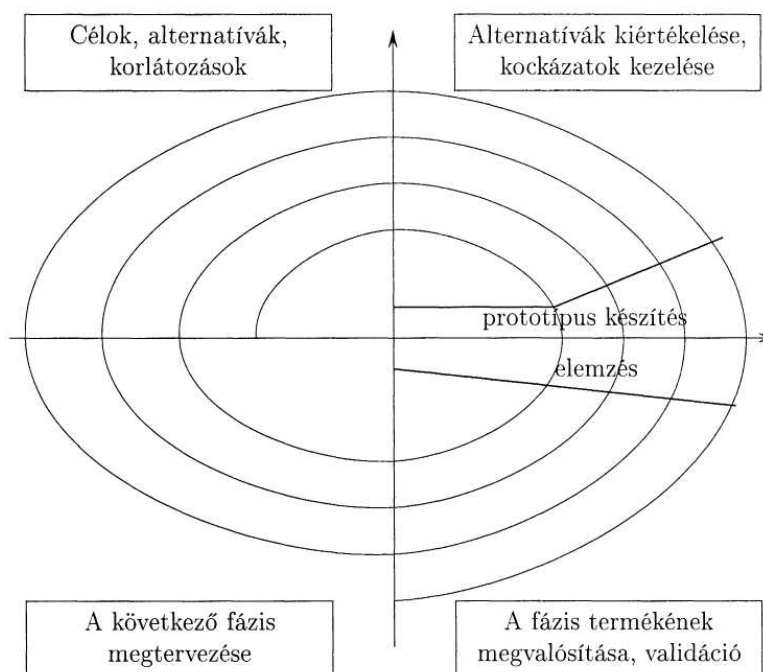
Hárányai:

- Nehéz a projekt áttekintése
- A gyors fejlesztés rendszerint a dokumentáltság rovására megy.

### 3. Boehm-féle spirális modell

Ez a modell egy iterációs modell. Az iteráció a spirális egy fázisával modellezhető, amely négy szakaszra bontható:

- Célok, utak, alternatívák, korlátozások definiálása
- Kockázatelemzés, stratégia kidolgozás
- Feladat megoldása, validáció
- Következő iteráció megtervezése



5. ábra. Evolúciós modell

Hárányai:

- A modell alkalmazása általában munkaigényes, bonyolult feladat.
- A projekt kidolgozásához szükséges szakembereket nem könnyű gazdaságosan foglalkoztatni.

## Az objektumelvű modellezés nézetrendszerei

**Objektumelvű programozás** = adatabsztrakció + absztrakt adattípus + típusöröklődés

- Absztrakció

Programozás adott szintjén a megoldás szempontjából lényegtelen részletek elhanyagolása.

- Adattípus

Az adattípus egy  $(A, F)$  rendezett pár, ahol  $A$  az adatok halmaza  $F$  pedig a műveletek véges halmaza.

$$\forall f \in F | f : A^n \rightarrow A$$

Létezik egyszerű és összetett adattípus.

**Típusosztály:** A típus komplex leírása, mely az adott adattípus absztrakt (PAR + EXP) és konkrét (IMP + BODY) leírását szolgálja, azaz:

Típusosztály = (PAR, EXP, IMP, BODY), ahol:

PAR = <paraméterek tulajdonságai >

EXP = <típusobjektumok halmaza és műveletei neve, szintaktikája, szemantikája >

IMP = <más osztályból átvett szolgáltatások >

BODY = <típusosztály ábrázolása, megvalósítása >

- Típusöröklődés

A típusöröklődés két fő formája: *specializáció* és *újrafelhasználás*

1. A subclass átveszi az absztrakt tulajdonságokat és azt az export részben használja fel
2. Típushalmaz, paraméterhalmazok, műveletek nevei átdefiniálódhatnak.
3. subclass típushalmaza = superclass típushalmaza

A specializáció következményei:

- Polimorfizmus

Minden változónak két típusa van: *statikus* (deklaráció során kapott) és *dinamikus* (deklaráció pillanatában megegyezik a statikussal, de később megváltozhat, ha egy superclass példánynak adunk értékül egy subclass példányt)

- Dinamikus kötés

A dinamikus típusnak megfelelő kiszámítási szabály hozzárendelése a függvényhez attribútumhoz, a végrehajtás pillanatában

## Nézetrendszerek

- Használati szempont

Kinek nyújt a rendszer szolgáltatást? (Személyek vagy más rendszerek, programok)

- Szerkezeti struktúráis, statikus szempont

Milyen egységek vannak, ezeknek mi a feladata, és hogyan kapcsolódnak egymáshoz?

- Dinamikus szempont

Az egyes részegységek hogyan viselkednek, milyen állapotokat vesznek fel, azokat milyen események hatására váltják? Milyen az egységek között együttműködés mechanizmusa? Időben hogyan játszódnak le közöttük az üzenetek?



- Implementációs szempont  
Milyen szoftverkomponensek, és azok között milyen kapcsolatok vannak?
- Környezeti szempont  
A rendszer milyen hardver és szoftver erőforrást igényel a megoldás során?

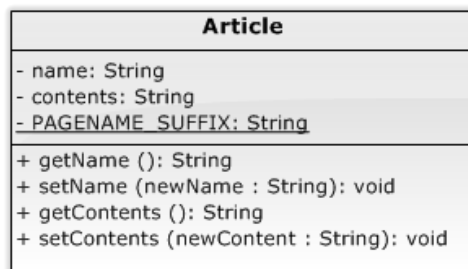
## Statikus modell (osztálydiagram, objektumdiagram)

### Osztálydiagram

A megoldás szerkezetét leíró összefüggő gráf, melynek csomópontjaihoz az osztályokat, éleihez pedig az osztályok közötti relációkat (öröklődés, asszociáció, aggregáció, kompozíció) rendeljük. A rendszerhez csak egy osztálydiagram tartozik.

### Osztályok

Egy osztály a következőképp néz ki:



6. ábra. Osztály

Az osztályt leíró téglalap három részre van osztva.

- Az első részbe az osztály neve kerül.  
Ha az osztály absztrakt, a nevét dőlt betűvel írjuk.
- A második részbe az osztály attribútumai kerülnek.

Az attribútum formátuma: *Attribútumnév : Típus*

A statikusságot aláhúzással jelöljük.

Az attribútumok láthatóságát is fel lehet tüntetni:

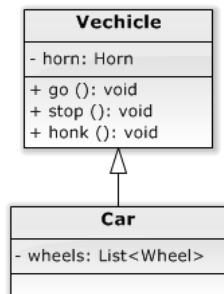
- *publikus (+)*
- *privát (-)*
- *védett (#)*
- A harmadik részbe az osztály metódusai kerülnek.

A metódusok formátuma: *Metódusnév(Paraméterlista):Visszatérési érték*, ahol a paraméterlista *Paraméternév:Típus* formátumú paraméterekből áll. Absztraktságot, statikusságot, és láthatóságot az előzőekben leírtakkal azonosan jelöljük.

## Osztályok közötti kapcsolatok

- Öröklődés

Két osztály közötti absztrakciós kapcsolatot jelöl



7. ábra. Öröklődés

- Asszociáció

Ez a legáltalánosabb reláció két osztály között. Az asszociáció két osztály közötti absztrakt reláció, amely kétirányú társítást fejez ki. A reláció absztrakt volta azt jelenti, hogy a reláció konkretizálása osztályok objektumainak összekapcsolásában valósul meg.

Az asszociációnak lehet:

- Neve, azonosítója
- Iránya
- Multiplicitása  
Akár egy érték, akár intervallum. A \* szimbólum kitüntetett szerepet kap, jelentése: bármennyi, akár nulla is. (Pl: 3, 1..4, 5..\*, \*)
- Szerepe
- Navigálhatósága

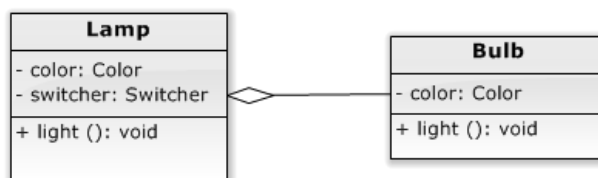
A társított osztályok közül csak az egyik ismeri a másikat. (Ha nem tüntetjük fel, kölcsönös elérhetőséget feltételezünk)



8. ábra. Asszociáció

- Aggregáció

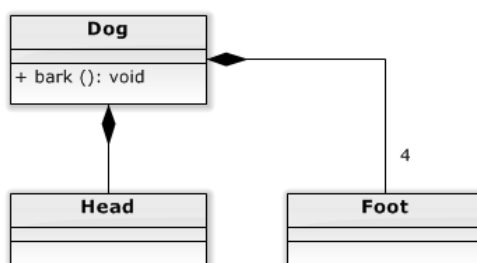
Az aggregáció egy speciális asszociáció, mely egész-rész kapcsolatot fejez ki. Azonban ha két osztály között aggregációs reláció áll fenn, a két osztály objektumai egymástól függetlenül is létezhetnek (Ezt ún. laza tartalmazási relációnak nevezik) A relációt jellemzi ezen kívül a tranzitivitás, és a közös attribútumok illetve szolgáltatások. Különböző aggregátumoknak lehetnek közös komponenseik.



9. ábra. Aggregáció

- Kompozíció

A kompozíció egy speciális aggregáció, mely *fizikai* tartalmazást jelöl. Nem jellemzi többé az objektumok független létezése, a két objektum egyszerre jön létre és szűnik meg. Tehát a tartalmazó objektumnak gondoskodnia kell a tartalmazott létrehozásáról és megszüntetéséről. Egy komponens legfeljebb egy tartalmazó objektumhoz tartozhat. A kompozíciós kapcsolat és az attribútum jellegű kapcsolat két objektum között szemantikailag azonos.



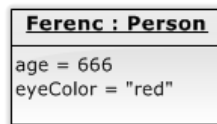
10. ábra. Kompozíció

## Objektumdiagram

Az objektumdiagram egyszeresen összefüggő gráf, amelynek csomópontjaihoz az objektumokat, éleihez pedig az objektumok közötti összekapcsolásokat rendeljük.

A rendszerhez különböző időpillanatokban más-más objektumdiagram tartozhat. (Viszont mindegyiknek meg kell felelnie az osztálydiagramnak)

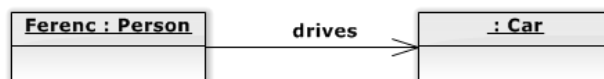
- **Objektumok** Az objektumokat az osztályokhoz hasonlóan egy téglalap írja le. Egy ilyen téglalapnak két része van. Az első részben az objektum neve (opcionális) és típusa található a következő formátumban: *Objektumnév : Típus*, melyet aláhúzással tarkítunk. A második részbe az objektum attribútumai és azok értékei kerülhetnek, a következő formátumban: *Attribútumnév=érték*.



11. ábra. Objektum

- **Objektumok közötti kapcsolatok** Az objektumokat összekötő relációk az osztálydiagramon lévőekkel megegyezők (Öröklődésnek ezen a szinten nincs értelme):

- **Asszociáció**



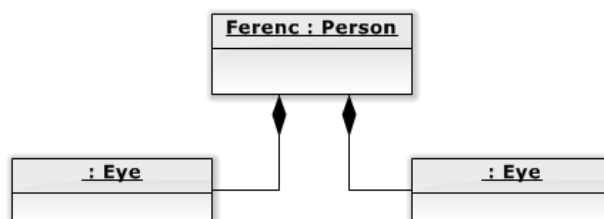
12. ábra. Asszociáció

- **Aggregáció**



13. ábra. Aggregáció

- **Kompozíció**



14. ábra. Aggregáció

# Dinamikus modell (állapotdiagram, szekvenciadiagram, együttműködési diagram, tevékenységdiagram)

## Állapotdiagram

Az állapotdiagram egy összefüggő irányított gráf, amelynek csomópontjaihoz az állapotokat rendeljük, éleihez pedig az eseményeket. (Két csúcs között több állapotátmenetet is jelölhetünk, hiszen több esemény hatására is létrejöhet)

- **Állapot:** Az objektum állapotát az attribútumok konkrét értékeinek n-esével jellemezzük.

Az állapotnak van azonosítója, mely legtöbbször az állapot neve (de lehet maga az invariáns, vagy az attribútumok konkrét értéke).

Az állapotot esemény hozza létre és szünteti meg. Az állapot mindaddig fennmarad, míg az attribútumok kielégítik az állapotot leíró invariánst. Az állapotot egy lekerekített téglalappal jelöljük, melyben az azonosítót tüntetjük fel.

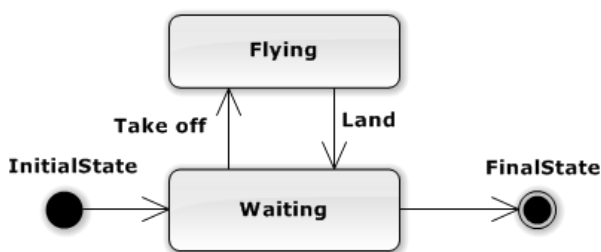
Speciális (rendszeren kívüli) állapotok: Kezdőállapot, Végállapot

- **Esemény:** Eseménynek nevezzük azt a tevékenységet, történést, amely valamely objektum állapotát megváltoztatja.

Az esemény lehet paraméteres vagy paraméter nélküli, és lehet előfeltétele. Az események között sorrendiség áll fent, így egy esemény lehet megelőző eseménye. Egy eseményt a következőképp írhatunk le:

<esemény>(<paraméterek>)[<feltétel>]/<megelőző esemény>

Az eseményeket az állapotok közötti állapotátmenetekre írjuk.

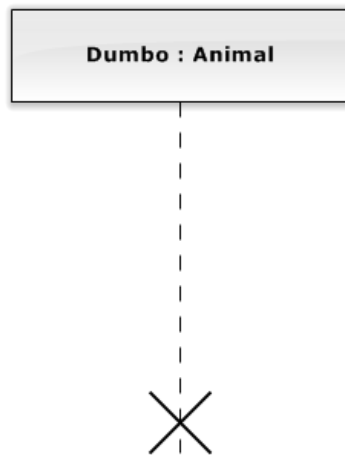


15. ábra. Repülőgép állapotgépe

## Szekvenciadiagram

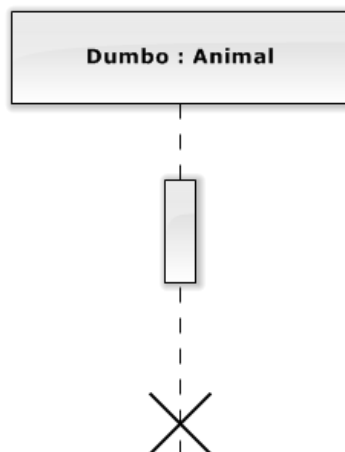
A szekvencia diagram az objektumok közötti üzenetváltások időbeli menetét szemlélteti.

- **Osztályszerp:** Az osztály szerepét olyan egy vagy több objektum testesíti meg, melyek az üzenetküldés szempontjából konform módon viselkednek.
- **Osztályszerp életvonal:** Az életvonal az osztályszerp időben való létezését jelenti.



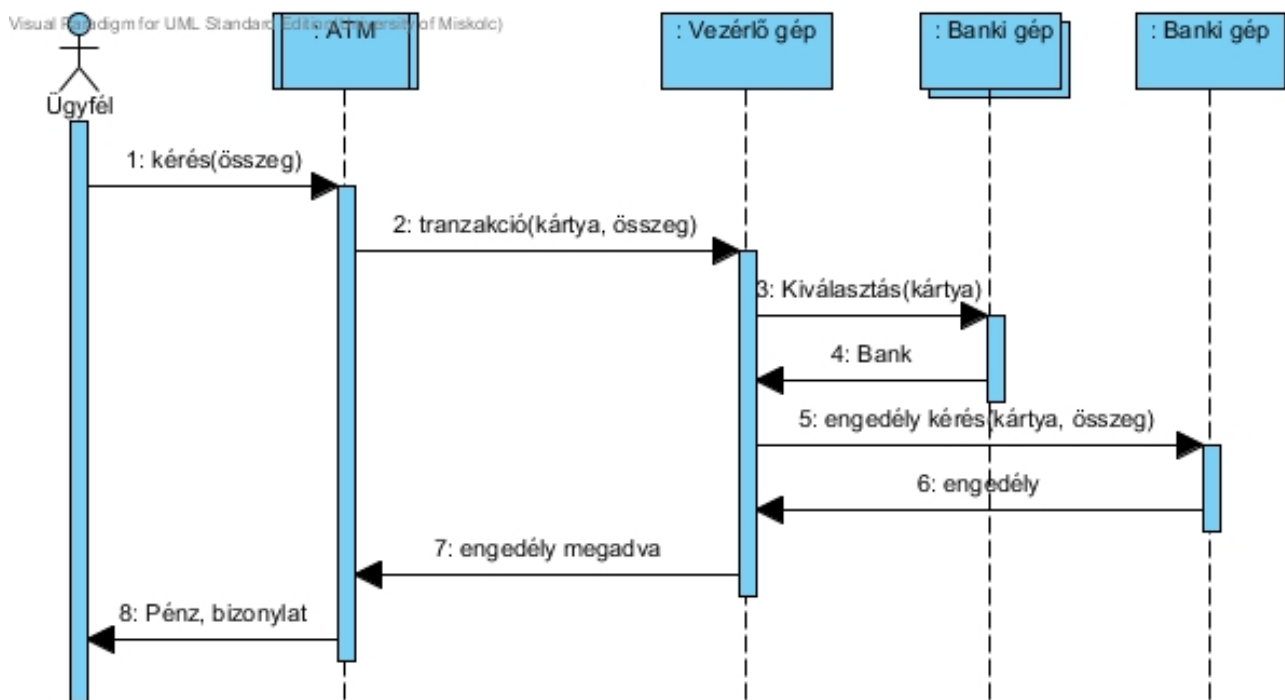
16. ábra. Életvonal

- **Aktivációs életvonal:** Az aktivációs életvonal azt az állapotot jelenti, amikor az osztályszerep megtestesítői műveleteket hajtanak végre, vagy más objektumok vezérlése alatt állnak.



17. ábra. Aktivációs életvonal

- **Üzenet** Az üzenet az objektumok közötti információátadás formája. Az üzenet küldésének az a célja, hogy az objektum működésbe hozza a másik objektumot. Az üzenet azok között az objektumok között jöhet létre, amelyek az objektumdiagramban kapcsolatban állnak. Az üzenetnek van azonosítója (neve, szövege), lehet paramétere, sorszáma.



18. ábra. Szekvencia diagram

## Együttműködési diagram

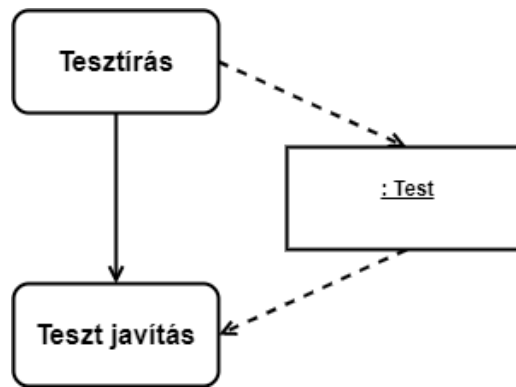
Az együttműködési diagram azt hivatott bemutatni, hogy miként működnek együtt az osztályok objektumai, milyen üzenetek cseréje révén valósul meg ez az együttműködés. (Csak azok az objektumok relevánsak, amelyek osztályait az osztálydiagramban asszociációs kapcsolat köt össze. A diagram mutatja ezt az összekapcsolást és az ehhez tartozó üzenetváltásokat, ezért az együttműködési diagram az objektumdiagram bizonyos értelemben vett kiterjesztésének tekinthető.)

Az üzenet küldését egy nyíl mutatja, amely az asszociáció mellett kap helyet és a címzett irányába mutat. Az üzenet azonosítója a nyíl mentén helyezkedik el. Az üzenetnek lehet argumentuma és eredménye. Ezeket egy kis körből induló nyíl mellett helyezzzük el, ahol a nyíl az információ áramlásának irányát mutatja.

## Tevékenységdiagram

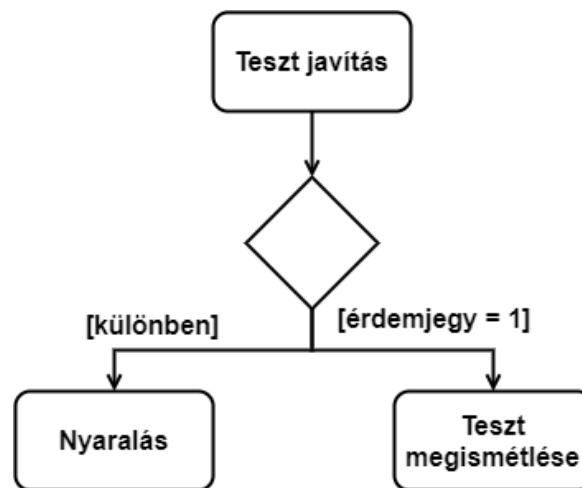
A tevékenységdiagram (aktivációs diagram) a probléma megoldásának lépéseit szemlélteti, a párhuzamosan zajló vezérlési folyamatokkal együtt.

Ha egy tevékenységet egy másik tevékenység követ közvetlenül, akkor a két tevékenységet nyíllal kötjük össze. Ha adatot (objektumot) ad át egy tevékenység egy másik tevékenységnek, akkor a küldő tevékenységből szaggatott nyíl vezet az objektumot reprezentáló téglalaphoz, és a téglalaptól szaggatott nyíl mutat a fogadó tevékenységre. A téglalapban szögletes zárójelek között megadhatjuk az objektum állapotát, státuszát is.



19. ábra. Objektum átadás

Lehetőség van arra, hogy bizonyos feltételek teljesülése esetén eltérő tevékenységeket hajtsunk végre, illetve tevékenységek végrehajtását feltételekhez kössük. Ekkor egy rombuszt kell elhelyeznünk a diagramban, amelyből kivezető nyilakra írjuk a feltételeket.



20. ábra. Feltétel ábrázolása

## Használati esetek diagramja

A használati esetek diagramja a felhasználók szempontjából kívánja szemléltetni azt, hogy a rendszer miként működik, függetlenül attól, hogy a szolgáltatásait hogyan valósítja meg.

A diagram részei:

- használati esetek (a rendszer funkciói)
- aktorok (a rendszeren kívüli, azzal interakcióba lépő egységek)
- relációk (két használati eset, vagy egy aktor és egy használati eset között állnak fent)

A használati esetek a rendszer funkcióinak összefoglalásai, szolgáltatási egységek. Ez az egység az akcióknak egy olyan sorozata, amelyekkel a rendszer a felhasználók egy csoportjával működik együtt.



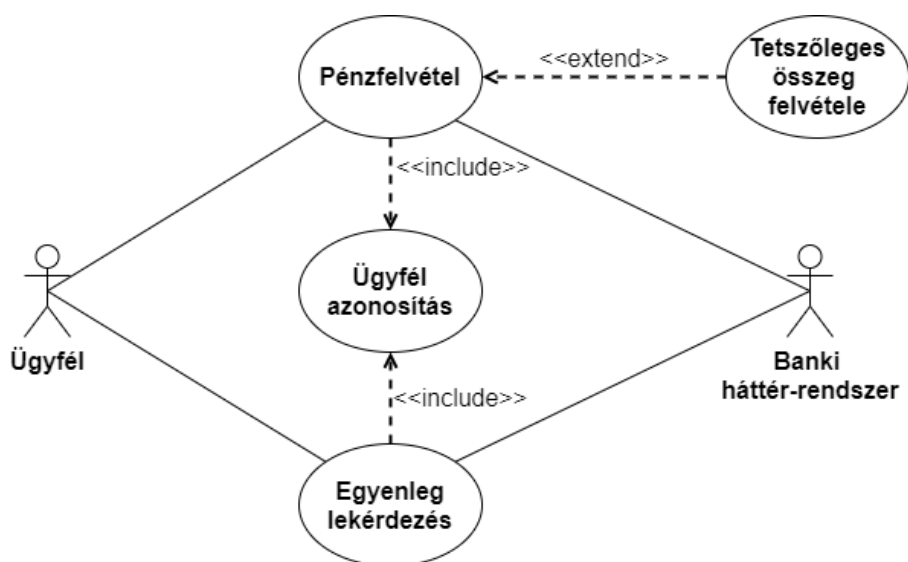
A használati esetet egy ovális alakzattal jelöljük. A használati eseteket téglalapba foglaljuk, ez jelzi a rendszer határait.

A felhasználók az adott rendszeren kívüli egységek, más programrendszerek, alrendszerek, osztályok, illetve személyek lehetnek. Ezek aktor szerepet töltenek be. A diagramon egy pálcikaember figurával jelöljük.

A felhasználási relációk kapcsolják össze a használati eseteket a felhasználókkal. A relációk egymással is kapcsolatban állhatnak, amit a diagramban fel lehet tüntetni.

A lehetséges relációk a következők:

- **Asszociáció:** Egy felhasználó és egy használati eset közötti kapcsolatot jelez. (Egyszerű vonal)
- **Általánosítás:** Az egyik használati eset a másik általánosabb formája. (Egyszerű vonal, végén fehér háromszöggel)
- **Kiterjesztés:** Az egyik használati eset a másikat terjeszti ki. Ennek során viselkedéseket illeszt be megadott beszűrési pontoknál. (Szaggatott nyíl `<<extend>>` felirattal)
- **Tartalmazás:** Az egyik használati eset tartalmazza a másik viselkedését. (Szaggatott nyíl `<<include>>` felirattal)



21. ábra. Használati esetek diagramja

Az ügyfél akár pénzt vesz fel, akár az egyenlegét kérdezi le az ügyfél azonosítás mindkét folyamatnak része (`<<include>>`). Az automata előre megadott összegeket ajánl fel kivételre, de tetszőleges, az ügyfél által megadott összeg is kivehető (`<<exclude>>`).