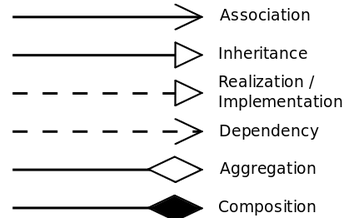
1. **Szoftver projekt fejlesztés lépései**

* **Cél kitűzése:**
  + miért csináljuk a projektet, mit szeretnénk elérni a fejlesztéssel
* **Kockázat elemzés:**
  + nagyobb projektet esetén már érdemes megvizsgálni előre, hogy milyen problémákba, akadályokba üzközhetünk, és milyen megoldásokat adhatunk azokra
* **Megtérülési vizsgálat:**
  + megint csak nagyobb, költségesebb projektek esetén lehet szükség megvizsgálni, hogy a befektetés mikorra térül meg
* **Erőforrások projekthez rendelése:**
  + Felelősök kinevezése és feladatkörök meghatározása, pl. projektvezető, tanácsadók, fejlesztők
* **Ütemterv elkészülés:**
  + Rögzítjük az olyan alapvetéseket, mint a határidő, munkaelosztás, stb.
* **Gyártás és Tesztelés**
* **Modellek: vízesés modell, V modell, (scrum?)**

1. **Követelmény specifikáció**

* néha nehéz leírni egy rendszer elvárt működését, így csak szolgáltatásokat, üzemeltetési és fejlesztési megszorításokat írunk le
* a követelmények tervezése és specifikációja egy kritikus szakasz
* a követelményeket lehetnek:
  + **felhasználói követelmények**: magas szintű absztrakt követelmények, ez a menedzserek és ügyfelek számára készül, akik nem rendelkeznek technikai ismerettel a rendszerről, leírja milyen szolgáltatásokat várunk el a rendszertől, és milyen megszorításokkal kell működnie
  + **rendszerkövetelmények**: a rendszer funkcióit, szolgáltatásait és működési megszorításait írja le, pontos kell legyen, meg kell határozni pontosan mit kell implementálni
* egy másik megközelítés a funckiónalitás szempontjából történik:
  + **funkcionális követelmények**: rednszerfunkciók ismertetése, hogyan reagáljon a rendszer bizonyos bemenetekre
  + **nemfunkcionális követelmények**: a rendszer tulajdonságaira vonatkoznak, magukba foglalják az időbeli és a fejlesztési folyamatra tett megszorításokat
  + **szakterületi követelmények**: a szakterület jellegzetességeti és megszorításait tükrözik

1. **UML diagramok. Használati eset diagram. Osztály diagram**

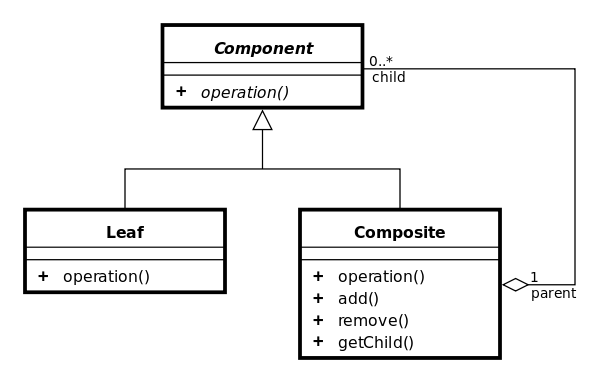
* egy általános célú modellező nyelv
* a rendszerek modelljének vizuális dokumentálására alkalmas eszköt
* a rendszer lehet szoftver, de akármilyen más típusú rendszer is
* a diagramok mellett találhatunk táblázatokat, illetve szöveges formákat is
* több diagram is vonatkozhat ugyanazon modellre, átfedhetik egymást
* a struktúradiagram központi diagramja az **osztálydiagram**, minden egyéb ebből származtatható
* a viselkedési diagramoknál nem határozható meg egy ilyen ősdiagram
* az interakciós diagramoknak három típusa létezik: idődiagram, kommunikációs és szekvencia
* a **Use Case** diagram célja leírni a modellezendő rendszer és környezete kapcsolatait
* egy funkcionális diagram, tehát a végrehajtandó funckiókat írja le, megmutatja kik és mire akarják használni a rendszert
* elemei: *rendszer, aktor, használati eset, relációk*
* az **Osztálydiagramok** az osztályok és a köztük lévő kapcsolatok ábrázolására használt modellezés
* szimbolikája:
  + Osztály szimbóuma, **láthatóságok: -** privát, # protected, + public
  + Kapcsoatok:
    - **asszociáció**: kétirányú összeköttetés, az objektumok között van kapcsolat
    - **származtatás**
    - **implementálás:**
    - **függőség**
    - **aggregáció**:tartalmazás, viszont önállóan is létezhet
    - **kompozició**:tartalmazás, a tartalmazott önmagában nem létezhet

1. **Architekturális minták. Model-View-Controler architektúra. Előnyök és hátrányok**

* az architektrurális minta egy általános, újrahasznosítható megoldás egy gyakori problémára
* a design pattern-nél kicsit tágabb fogalom
* néhányukat alapértelmezetten tartalmazzák bizonyos framework-ok
* az egyik legismertebb az MVC
* az **MVC** lényege hogy elkülöníti az adatot a nézettől és a kontrollertől:
  + **modell**: az alkalmazás által felügyelt információkat kezeli, illetve az ehhet stükséges ligikát
  + **nézet**: megjeleníti a modellt egy megfelelő ablakban, mely alkalmas a felhasználói interakcióra, különböző célokra különböző nézetek vannak ugyanahhoz a modellhez
  + **vezérlő**: az eseményeket, a felhasználói műveleteket dolgozza fel és válaszol rájuk, illetve a modellben történő változásokat is kiválthat
* előnye, hogy teljesen szétválasztja a részeket, könnyebben tesztelhetőek, fejlesztők a specialitásukkal foglalkozhatnak
* hátránya kissebb projekteknél a sok munka, illetve több időt vesz igénybe a betanulás, már meglévő projektet nehéz átalakítani

1. **Tervezési minták. Composite, Singleton és Observer minták**

* a tervezési minták a programozásban felmerülő gyakori problémákra adnak egy általános megoldást
* általában egymással együttműködő objektumok és osztályok leírása
* nem mutatnak kész tervet, inkább csak egy sablont
* **Composite**
  + olyan minta, mely azt írja le, hogy az objektumok egy csoportját ugyanúgy kell kezelni, mint az adott objektum példányait külön-külön
  + akkor használjuk ha több objektumot akarunk azonos módon használni, és majdnem azonos kódjuk az egyes elemek kezelésére



* **Singleton**:
  + egy objektumra korlátozza egy osztály létrehozható példányainak számát
  + minden konstruktora privát kell legyen, illetve bizonyos operátorok is
  + osztályszintű metódus jelenti a megoldást
* **Observer**:
  + olyan szoftvertervezési minta, melyben egy objektum, melyet alanynak hívunk, listát vezet alárendeltjeiről, akiket megfigyelőknek hívunk és automatikusan értesíti őket bármilyen állapotváltozásról
  + többnyire elosztott eseménykezelő rendszerek kialakításakor használjuk
  + memóriaszivárgást okozhat

