## Követelménymodellezés UML segítségével

### Bevezető gondolatok

Modellen a valóságban létező, vagy a jövőben elkészítendő dolog, tárgy, személy, szervezet, valamilyen más rendszer, vagy akár egy fogalom egyszerűsített vázlatos mását, vagy leírását értjük. Egy olyan minta, ami segítségével elkészít. A modellek a legkülönfélébb meg helyeken és formában jelenhetnek meg, lehet az egy ház tervrajza, vagy egy makett, vagy egy térkép. A modellek lehetnek matematikai, szöveges, vagy képi megjelenésűek is. A követelmények, és más szoftverfejlesztéssel kapcsolatos tevékenységet gyakran modellezünk képi eszközökkel, erre a vizuális modellezésre egy elterjedt eszköz az UML. A követelményfeltárás során általában modellezhetünk üzleteket, üzleti folyamatokat, üzleti szabályokat, üzleti rendszereket, alkalmazásokat rendszerfelépítéseket, egy rendszer elemeit, és azok egymásra hatását. Maga a modellezés nagyban segíti, a probléma megértését és a kommunikációt a fejlesztésben érdekeltek között. A szoftverfejlesztési gyakorlatban általában több szinten, több absztrakción is modellezünk, kezdvén a nagy távlati kis részletességű modellektől, a nagyobb részletességűek felé haladva, majd elérve végül a program megvalósításának szintjét. Fontos észben tartani viszont, hogy nem csupán a modellezés kedvéért modellezünk, az ilyen terveink legyenek a gyakorlatban kivitelezhetőek, és az elemzésükkel töltött idő is maradjon ésszerű korlátok között, ne menjen a fejlesztés kárára. A magas szintű modelljeinket alkossuk meg olyanra, hogy azt akár a rendszer felhasználója is megérthesse. A rendszer az esetek túlnyomó többségében embereknek készül, és egy üzlet érdekeit szolgálja, ezt pedig érdemes a fejlesztés minden szakaszában, észben tartani.

### Az UML bemutatása

Az UML előtt számos Objektum Orientált modellezési eszköz terjedt el, melyek megnehezítenék a különböző eljárást és szemléletet alkalmazó fejlesztők kommunikációját. A 90-es évek közepén a Rational cég felfogadta az OMT metodikát kifejlesztő James Rumbaugh-t, A Booch metódus atyját Grandy Booch-t, majd klésőbb az Objectory kifejlesztőjét Ivar Jacobson-t, hogy alkossanak meg egy átfogó és egységes modellezési módszert. Együtt megalkották a UML-t, azaz az egységesített modellező nyelvet. Mivel a három legelterjedtebb módszer kifejleszője alkotta meg, hamar De Factó, majd miután az OMG (Object Management Group) átvette a felügyeletét, 1997-re az UML 1.1-es változata De Jure szabvány lett.

Mint ahogy a neve is mutatja az UML egy nyelv, a modellezés eszköze, amely rendelkezik nyelvi elemekkel, és szabályokkal, de a fejlesztésre vonatkozóan nem ad ajánlásokat, így nem fejlesztési metodika. Az UML egy olyan nyelv, mely támogatja a modellünk egységes elemekből való felépítését. Egy közös nyelv lehet, mely összeköti az üzleti elemzőket, a szoftverfejlesztőket, a tervezőket, a tesztelőket, az adatbázis-tervezőket és más a fejlesztési projectben résztvevő szakembereket. Napjainkban kiemelt fontosságú, a résztvevők átfogó és egyértelmű kommunikációja, melyre egy ilyen szabványosított nyelv kiválóan alkalmas lehet.

Modellek a valóságban is előforduló entitások és rendszerek vázlatos másai. A modell lehet szöveges matematikai, vagy akár kézzel fogható, mint egy makett. Az UML modelljeink megalkotása vizuális eszközökkel történik és a modellünk különböző nézeteit diagramok adják.

A modellező nyelv felfogható egy eszköztárként, mely számos diagram típust, és azokhoz számos elemet biztosít. Ezek közül kiválaszthatjuk az éppen a problémánknak, és a rendszer éppen feltérképezendő vetületének leginkább megfelelő eszközt, azaz diagram típust.

Modellezés közben a modellezni kívánt rendszert számos szinten, a távoli absztrakt leképzéstől, az egészen részletes modellekig haladva ábrázoljuk. Az egyre finomabb és finomabb részletességű modellek kifejlesztése során megértjük a kifejlesztendő rendszert, a rendszert alkotó elemeket, és azok kapcsolatait.

Az UML modelltípusai, elterjedt modellezési szintjei, az üzleti modell, a követelménymodell, a felépítésmodell, az alkalmazási modell, és az adatbázis modell. Az üzleti modell, különböző üzleti folyamatokat, munkafolyamatokat, és üzletszervezési lépéseket tartalmazhat. A követelménymodelleket a rendszertől elvárt szükséges működés analizálására és rögzítésre, illetve azok megrendelővel való kommunikációjára használhatjuk. A felépítésmodell alkalmas a rendszer magas szintű áttekintésére, a rendszer különböző részei közti kommunikációjának vizsgálatára, illetve a tervezők és a fejlesztők közti kommunikáció támogatására. Alkalmazási modell már a rendszeren belüli alacsonyabb szintű felépítést hivatott modellezni. Az adatbázis modell az adatbázis szerkezetének és az alkalmazásokkal való együttműködésének megtervezését hivatott segíteni. Ezeken a modellezési szinteken különböző modelltípusokat használunk.

Az UML a diagramokat két nagy csoportra bontja, vannak, amik a modellezni kívánt rendszer struktúráját képesek leírni, és vannak olyanok mely a rendszerben lezajló dinamikus viselkedéseket mutatják. Ezeket a csoportokat szokták szerkezeti, illetve viselkedési nézetnek is nevezni.

A strukturális diagramok közé sorolható az osztály diagram, a komponens diagram, az összetett struktúradiagramok, a telepítési diagramok, az objektumdiagramok és a csomagdiagramok. A viselkedési diagramok közé pedig az aktivitásdiagram, az állapotgép diagram a Use-Case, azaz a használati eset diagram, a szekvencia diagram az interakciós diagram Kommunikációs diagram, az interakció-áttekintő diagram és az időzítés diagram tartoznak.

A hatékony modellezés érdekében érdemes betartani néhány szabályt. Az egyik ilyen szabály, hogy egy-egy modell elem a modellen jobb, ha egyedi, de a modell különböző nézetében, különböző diagramokon és diagram típusokon jelenhet meg. Tehát a modell tartalmazza a rendszer összes elemét, de egy elem megjelenhet több diagramon is.

Fontos továbbá, hogy az üzlet és a rendszer változását ezek a modellek is lekövessék, folyamatosan frissüljenek, hogy nehogy egy inkonzisztens, a valóságtól távol álló állapotot mutassanak. Az inkonzisztens állapotot mutató modellek számos félreértést okozhatnak a fejlesztés, és a kommunikáció során. A modell egyes szintjeinek és a szoftverrendszer forráskódjának szinkronban tartására számos eszköz létezik. Amikor a modellből hozzuk létre a kódot, kódgenerálásnak hívjuk, és az a folyamat melyben a kód alapján hozzuk létre a modellt a reverse engeneering.

A szabványos UML nyelv számos eszközt kínál a legkülönbözőbb rendszerek modellezésére, mindemellett módot ad az eszköztár kiegészítésére, vagy a diagramjaink extra információval való ellátására. Talán a legegyszerűbb ilyen módszer, mellyel grafikailag nem, vagy csak nehezen jelölhető információt fűzhetünk az elemeinkhez, a megjegyzés, mely a diagramon egy behajtott sarkú lapként jelenik meg. A megjegyzések egyszerű szöveges leírás mellett tárolhatnak formalizált megszorításokat. A megszorításokat kapcsos zárójelek közé írt formális képletek, megadásukra az OCL (Object Constraint Language) egy elterjedt eszköz. A megszorítás egy feltételt határoz meg az elemen. Egy speciális válfaja a kulcsszavas érték, mely az elemen név-érték párokként jelöl különböző információkat.

A szöveges megszorítások mellett a másik elterjedt modellkiterjesztési metódus a sztereotípia, mely segítségével új modellelemek felvételére és a meglévő elemek magas szintű tipizálása használatos. Formális jelölése francia-idézőjelek között a sztereotípia neve által történhet. Esetenként a sztereotípiával ellátott elem a diagramon egyedi megjelenést is kaphat.

A sztereotípiáink és megszorításaink tárolására az UML profilok szolgálnak. Különböző szakirányokhoz és fejlesztői platformokhoz és környezetekhez számos előre definiált profil létezik.

Az UML alapú modell eszközök közti szabványos kommunikációjára az XMI (XML Metadata Interchange) formátumot ajánlja az OMG. Ez a XML (Extensible Markup Language) alapú leíró nyelv képes tárolni az UML metamodelljét. A metamodell az UML modell egyfajta modellje, mely meghatározza az UML-ben használható elemeket, azok tulajdonságait, lehetséges kapcsolataikat, és a kapcsolatotok jelentését. A metamodell határozza meg a modellezés szabályait, és az UML bővíthetőségének módjait is.

## A követelmények modellezése közben használt diagram eszközök

### A használati-eset diagram

A használati eset vagy más néven Use-Case diagram a rendszer felhasználóinak a szemszögéből tekintve ábrázolja a rendszer funkcióit, és céljait. A fejlesztendő szoftverrendszerben megjelenő követelmények áttekinthető ábrázolásának az egyik elterjedt eszköze. Az ábrázolás középpontjában a rendszer külső felhasználói és az általuk végezhető üzleti tevékenységek állnak.

Szereplőknek vagy aktoroknak hívjuk azokat a felhasználói köröket, melyek használni kívánják a rendszerünket. Az ilyen aktorok általában a valóságban létező felhasználói csoportokat vagy szerepköröket jelölnek, és gyakran megegyeznek a megrendelő szervezet egyes munkahelyi beosztásaival. Szereplőként szoktuk továbbá jelölni a fontosabb kapcsolódó külső rendszereket, illetve esetenként az olyan külső eseményeket, melyekre a rendszer reagál. A rendszer felhasználói a diagramon gyakran pálcika emberként jelennek meg, de számos eszköz módot ad a megjelenés testre szabására, ezzel javítva a diagram kifejező erejét. A szereplők a modellben <<aktor>> sztereotípiájú elemek.

A szereplők által végezhető feladatokat, üzleti célokat és üzleti tevékenységeket nevezzük használati esetnek, vagy az angol terminológiát átvéve Use-Case-nek. A diagramon megjelenő használati esetek többsége a kifejlesztendő rendszer későbbi elvárt szolgáltatása, azaz a rendszer kifele mutatott kapcsolódási pontja. Emellett megjelenhetnek más a rendszer vagy az üzlet szempontjából fontos külső folyamatok is, külső használati esetek formájában. Minden használati eset teljes forgatókönyvvel azaz scenarióval kell, hogy rendelkezzen, ami megadja, hogy a szolgáltatás milyen lépésekből áll. A használati esetek ovális alakzatként jelennek meg. A rendszer felelősségi körébe tartozó használati eseteket érdemes, kerettel elválasztani a rendszert használó külső szereplőktől, és az esetlegesen megjelenő külső feladatoktól.

A szereplők és feladatok között akkor áll fenn kapcsolat, amikor az adott aktor és az adott használati eset között kommunikációs és vagy utasítási kapcsolat van. Általában a kommunikációs kapcsolatot egyszerű az aktort a használati esettel összekötő asszociációt kifejező egyenes vagy, ha úgy tetszik, irányítatlan nyíl fejezi ki. A gyakorlatban gyakran jelöljük nyílheggyel, hogy az adott szereplő aktív munkát végez a rendszeren, vagy passzív, tehát a rendszer nyújt számára valamit, és az irányítatlan nyilat meghagyjuk arra az esetekre, amikor ténylegesen kétirányú kommunikáció folyik a rendszer és a szereplő között.

Az aktorok egymás között szabvány szerint nem lehetséges az asszociációs, azaz társítás kapcsolat, de a gyakorlatban a valóságot szimulálva jelölhetjük így a különböző aktorok közti a rendszer szempontjából fontosabb kommunikációt, párbeszédet vagy utasításváltást.

Az aktorok közti másik jellemzően használt kapcsolat az általánosítás-pontosítás viszony. Egy pontosított aktor rendelkezik általánosabb ősének a rendszerrel kapcsolatos összes képességével, és azt kiegészíti saját speciális, az őse által nem végezhető funkciókkal. Amennyiben két szereplő nagy mennyiségű közös kapcsolattal rendelkezik, jellemzően azonos szolgáltatásokhoz kapcsolódnak, érdemes lehet megvizsgálni, hogy nem-e állnak leszármazási kapcsolatban. A pontosítást háromszögben végződő nyílheggyel szemléltetjük, mely mindig az általánosabb fél felé mutat.

Feladat és feladat között számos kapcsolati mód értelmezett. Megjelenik itt is az általánosítás-pontosítás viszony. A pontosított, a használati eset, az általános ősének egy válfaja. Gyakran egy általános szolgáltatás leszármazottjai ősükéhez hasonló jellegű feladatok, de annak egy-egy konkrét válfajai. Jelölése az aktorok közötti leszármazás jelölésével azonosan, háromszögben végződő nyíl.

Egy másik lehetséges kapcsolat a használati esetek között a kiterjesztés kapcsolat, amikor az egyik Use Case a másik scenárióját opcionálisan bővítheti. Ez a bővítő mellékforgatókönyv a rendszer egyes állapotaiban hajtódik csak végre. Jelölhet abnormális, kivételes folyamatágat, de az is lehet, hogy egy bizonyos külső paraméter függvényében végrehajtandó rendszerfunkció. Ez a bővítő funkció több feltétel alapján is végbemehet, az ilyen feltételeket nevezzük bővítési vagy kiterjesztési pontnak. A kapcsolat jelölése egy <<extend>> sztereotípiával ellátott szaggatott nyíllal történik.

A beillesztett feladat is egy elterjedt kapcsolat, ahol az egyik feladat beékelődik a másik feladat lépéssorába. Ez a bővítéssel ellentétben kötelezően és csak egy pontban változtatja meg az alapszolgáltatás scenarióját. Gyakran élünk ezzel az eszközzel, ha hangsúlyozni szeretnénk egy szolgáltatás részforgatókönyvét, vagy ha egy részforgatókönyv több szolgáltatásnak is a része. A beillesztés jelölésére az <<include>> sztereotípiájú szaggatott nyíl szolgál.

Egy másik ritkábban használt kapcsolati mód a meghívás, ami azt fejezi ki, hogy az egyik folyamat kivált egy másik szolgáltatást. Ezt a kapcsolatot <<invokes>> sztereotípiájú szaggatott nyíl jelöli a diagramjainkon.

Fontos kiemelni, hogy az ilyen kapcsolatok megléte nem jelenti azt, hogy a diagramjainkon meg kell, hogy jelenjen a rendszerben fellépő összes elágazás, vagy az összes kis részfunkció, ami majd a későbbi részletesebb tervezési diagramjainkon és a végső forráskódban megjelenik. Itt csak a fontosabb szolgáltatásokat, részszolgáltatásokat és kivételes ágakat jelöljük.

### A tevékenység diagram

A tevékenység diagram, amit neveznek aktivitás diagramnak is, a rendszer időben lezajló változásainak a szemléltetésére szolgáló egyik eszköz. A használatával igyekszünk a rendszerben megjelenő üzleti munkafolyamatokat, illetve a rendszer tevékenységeinek lépéssorát, grafikusan modellezni.

A munkamenet kezdőpontját egy feketén kitöltött körlap szemlélteti, innen kezdjük a végrehajtást, ez nyíllal rámutat az első tevékenységre.

A munkamenet lépéseit művelet szimbolizálják, a terminológiában ezeket nevezik még tevékenységeknek, illetve aktivitásoknak is. Ezek lekerekített végű

A munkamenet végrehajtási irányát nyilakkal szabályozzuk. Ezek az egyszerű nyilak kötik össze a diagram megfelelő elemeit, ezzel meghatározva a lépések sorrendiségét.

Döntési pontok a munkafolyamokban megjelenő elágazásokat reprezentálja. Egy elágazásból egy feltétel kiértékelése alapján kettő vagy több irányba folytatódhat a munkafolyam. Az elágazás rombuszként jelenik meg, és a nyilakra írt úgynevezett őrzőfeltételek jelölik, hogy az adott irányba milyen feltétel teljesülése révén juthatunk el.

A diagramon jelölhető az is, hogy az adott műveletet ki végzi. Erre a feladatra úgynevezett úszósávokat vagy más terminológia szerint rekeszeket alkalmazunk, amik a diagramon téglalappal elkerített részek. A téglalap felső részén helyezkedik el a téglalapban megjelenő összes műveletének végrehajtását végző felelős szereplő vagy alrendszer neve.

Párhuzamos tevékenységek jelölésére is van mód, ilyenkor azt jelöljük, hogy az adott ágak végrehajtási sorrendje számunkra lényegtelen, és ha lehetséges, akár párhuzamosan is végbemehetnek. Jelölhetjük emellett a lépéssorok közti szinkronizációt is. Szinkronizáció után, a szinkronizált műveletsorokat végrehajtottnak tekinthetjük, mintegy megvárják egymást, hogy minden ág végre legyen hajtva. Ha nem használunk szinkronizációt, és egyszerűen egy műveleten egyesítjük a folyamatszálakat, akkor úgy tekintjük, hogy az először végrehajtott tevékenységsor megszakítja a többi végrehajtását. Az elágazás egy fekete kitöltésű téglalap, melyből nyilak mutatnak a párhuzamos lépéságak kezdőműveletére. A szinkronizálás, is egy ilyen téglalap, amire a szinkronizálandó lépéssorok nyilai mutatnak.

A folyamat végét egy fekete kör, centrumában egy feketén kitöltött körlap, mintegy célkeresztként jelöli. A munkafolyamatban akár több ilyen végjelző jel is szerepelhet.

### Sztereotípiákkal ellátott elemzési osztálydiagram

Az elemzésdiagram arra hivatott, hogy magas absztrakción mutassa a rendszerben megjelenő osztályokat, és a köztük fennálló kapcsolatokat. Ezen a szinten csak az osztályok nevei szerepelnek, és nincsenek feltüntetve az állapotokat tárolni képes attribútumok se a műveleteket végző konkrét metódusok. Az üzleti elemzésdiagram egy sztereotípiákkal ellátott elemzési osztály diagram, mely segítségével részletesen elemezhetjük a szereplők, és a rendszerben megjelenő további elemek statikus kapcsolatait. Ezeken a diagramokon már a fejlesztők szemszögéből modellezzük az üzletet megvalósító rendszert.

A gyakorlatban, ezen a modellezési szinten háromféle sztereotípiával látjuk el az elemeket. Az ilyen elemek lehetnek határoló-, irányító- vagy entitásosztályok. Ezek a diagramokon általában megjelenésükbe is jól elhatárolódnak.

A határoló osztályok hivatottak reprezentálni a felhasználói- vagy más rendszerekkel való összeköttetést biztosító interfészeket. Az adott osztály határoló osztály mivoltát a <<boundery>> sztereotípia adja.

A rendszerben feldolgozási és irányító szerepet betöltő osztályok a <<controller>> sztereotípiával jelölt kontroller osztályok.

A harmadik osztálytípus, mely leginkább adattároló szerepet tölt be az <<entity>> sztereotípiájú entitás

Az elemek között számos különböző kapcsolat lehet. Az egyik legáltalánosabb kapcsolat az asszociáció, mely azt szimbolizálja, hogy az egyik elem ismeri a másikat, képes vele kommunikálni, üzeneteket küldeni neki. Ez a társítás lehet egyirányú, melyet nyíllal jelölünk, vagy kétirányú kölcsönös ismerettség, melyet a két elemet összekötő egyenessel jelölünk. Lehetséges több elemet is azonos asszociációban elhelyezni, ilyenkor az asszociban résztvevő osztályok egy rombuszhoz kapcsolódnak, mely tárolja a kapcsolat nevét. A részletesebb osztálydiagramokon elképzelhető az is, hogy magához a kapcsolathoz tartozik osztály.

A függőség kapcsolat azt fejezi ki, hogy az egyik elem függ a másiktól, tehát a másik megváltozása magával vonhatja az egyik megváltozását.

Az aggregáció egy olyan társítás, ahol az egyik elem tartalmazhatja a másik elemet. Jelölése egy kitöltetlen rombuszban végződő nyíl, mely rombusza a tartalmazó felé néz.

A kompozíció egy olyan társítás, ahol az egyik elemnek része a másik elem, az ilyen elemek élettartama egymáséhoz kapcsolódik, nem létezhet az egyik a másik nélkül. Jelölése egy feketén kitöltött rombuszban végződő nyíl.

Az általánosítás itt azt jelenti, hogy van egy olyan elem, mely egy vagy több elemnél általánosabb vagy, hogy az egyik elem a másik elem pontosítása, tehát rendelkezik mindazokkal a felelősségi körökkel, mint az általános őse, és ezeket további rá jellemzően speciális funkcionalitásokkal bővíti ki. Jelölése egy háromszögben végződő nyíl, mely mindig az általánosabb elem felé mutat.

A kapcsolatoknak, és ez különösen igaz az egyszerű asszociációs társításokra megadhatunk a kapcsolatnak egy nevet, és elemenként azt, hogy a kapcsolatban mi az adott elem betöltőt szerepe.

A kapcsolatnak megadhatunk multiplicitást is, mely jelzi, hogy az adott elemekből mennyi vesz részt a társításban. Ez lehet egy az egyhez (1-1), egy a többhöz (1-n), több a többhöz(n-m) kapcsolat, illetve használható a csillag '\*' karakter annak a szimbolizálására, hogy az adott elemből 0, vagy nem meghatározható számú lehet. Az elem számosságára intervallum is megadható, a n..m (ahol n<m, és az m helyén állhat \* is) formában.

### Szekvencia diagram

A sorrend diagram a rendszer viselkedését írja le, méghozzá úgy, hogy a rendszer elemei között fellépő kölcsönhatások időbeli viszonyait állítja a modellezés középpontjába.

A diagram tetején vízszintesen helyezkednek el az osztály diagramokon felvett elemek közül azon osztályoknak az objektum-példányai, melyek az éppen modellezni kívánt rendszerviselkedésben szerepet játszanak. Az idő a diagram tetejétől az alja felé halad. Egy függőleges szaggatott életvonal jelzi, hogy az adott objektum létrejött, és a rendszerben jelen van. Az életvonal végén keresztben való áthúzás pedig azt fejezi ki, hogy az adott példány megszűnt.

Aktivációs vonal azt jelzi, hogy az adott időintervallumban az objektum aktívan tevékenykedett. Ezt egy fehér téglalap jelzi.

Az elemek közti üzenetváltások, amik a részletesebb szekvenciadiagramokon, már konkrét metódushívásokat jelölnek, a diagramon különböző nyilakként jelennek meg. Ezek az üzenetek befolyásolják az adott elemek aktivációs vonalát. A különböző üzenettípusokat különböző nyíltípusokkal szemléltetjük. Ezek a nyilak általában vízszintesen haladnak, de olykor szemléltethetjük a hosszabb időt igénylő, például hálózaton keresztül történő üzenettovábbítást a nyíl hegyének átlósan lefelé történő elmozdításával.

Az egyszerű üzenet azt fejezi ki, hogy az aktív objektum átadja a vezérlést egy másik példánynak, így ezután az válik aktívvá. Ez többnyire egy egyszerű eljáráshívást reprezentál, és jelölésére egy egyszerű nyilat használunk.

Visszatérési üzenettel jelezhet az elem, hogy visszaadja a vezérlést az őt aktiváló példánynak.

Szinkronizációs üzenetről beszélünk akkor, amikor az üzenet küldője blokkolt állapotba kerül amíg a fogadó nem fogadja az üzenetét.

Időhöz kötött várakozás azt fejezi ki, hogy a küldő várakozik t időegységet, majd ha addig nem kap visszajelzést a fogadótól, folytatja a munkáját.

Randevú üzenet azt fejezi ki, hogy a fogadó várakozik a küldőre.

Aszinkron üzenet hatására a küldő nem kezd el várakozni, hanem folytatja tovább a tevékenységét.

Saját delegálásnak nevezzük azt, amikor az adott elem saját magának küld üzenetet. Ezt az aktivációs vonal megduplázásával jelöljük.

A diagramon a különböző vezérlési szerkezetek, mint az elágazás vagy a ciklus is megjelenhetnek. Az elágazást többféle képen is jelölhetik. Egyik jelölési mód, hogy az elemek élet, és aktivációs vonalát egy szakaszon megduplázzák, és ezek felyezik ki a különböző elágazási ágakat. Egy másik megoldás beágyazott al-szekvenciadiagramokat használni a különböző ágak használatára. Esetenként az egyszerűbb elágazások egyszerű szöveges vagy OCL (Object Constrait Language) segítségével leírt megjegyzésben is jelölhetők. Az utóbbi két módszer használatos a ciklusok jelölésére is.

A diagram a feladatok sorrendjét, és időbeliségét nagyszerűen képes ábrázolni, de az elágazások, illetve a ciklikusságok jellemzésére, használható aldiagramok és megjegyzések átláthatatlanná tehetik. Ezek szemléltetésére másik eszközt lehet érdemes választani, például a tevékenység diagramokat. A szekvencia diagramokon továbbá inkább csak közelítőleg szemléltethető, a műveletek, vagy az üzenetek időigénye, a tervezés szakaszban egy-egy elem pontosabb időbeli állapotváltozásait szemléltethetjük időzítés diagrammal.

A szekvenciadiagramok akkor használhatóak hatékonyan, ha az adott tevékenységsorrend viszonylag kevés elem közti sűrű kommunikáció révén megy végbe.

### Együttműködési diagram

A szekvencia mellett egy másik a rendszerben megjelenő interakciókat mutató diagramtípus az együttműködési vagy más néven kommunikációs diagram. Itt viszont nem az időbeliség helyett itt hangsúlyosabb az objektumok szerveződése, és a kapcsolataik.

Az üzenetváltások, hasonló típusúak és hasonló célt szolgálnak, mint a szekvencia diagramoknál, de egymásutániságuk itt halványabban, egyszerű számozásként jelenik meg.

Ezen a diagramtípuson kényelmesebben szemléltethető, ha sok elem vesz részt az adott tevékenységben, de ezek között viszonylag kevés üzenetváltás zajlik.

### Állapotdiagram

A tevékenység diagram mellett, az állapot-átmenet diagram egy másik eszköz a rendszer időbeli változásainak a szemléltetésére, de az aktivitásokkal szemben itt sokkal inkább a rendszerben külső események hatására bekövetkező állapotváltozások állnak a modellezés középpontjában. Az állapot-átmenet diagramok nem az objektum orientált világból származnak, de jól illeszkednek az OO szemlélethez.

Az állapot egy rendszerelem, rendszerkomponens vagy objektum élettartama közben felvehető különböző helyzeteit szemlélteti. Az OO világban az állapotot a példányok, és objektumcsoportok attribútumainak az értéke határozza meg. A modellezés során egy-egy állapotot ezeknek az attribútumoknak egy-egy értékhalmaza, a vizsgálat szempontjából lényeges jellemzők csoportja jelöli ki. Így az állapot egy átfogóbb, absztraktabb képet ad az adott rendszerelemről, mintha csupán értékeket vizsgálnánk. Értelmezett két kitüntetett állapot, a kiindulási és a befejezési állapot, melyek jelölésére egy fekete körlap, illetve egy fekete célkereszt szolgál.

Az elem dinamikus változását az állapotátmenetek fejezik ki, melyeket események váltanak ki. Tehát az elemet érő külső események, olyan történések, melyek kiválthatnak a rendszerben egy állapotváltozást. A modellezni kívánt elem, a különböző állapotaiban különbözőképpen reagálhat egy adott külső eseményre, vagy akár teljesen figyelmen kívül is hagyhatja azt. Az esemény egy rövid időpillanat alatt játszódik le, és jelölésére egy a kezdőtől a végállapotba mutató nyíl, és rajta az esemény neve. Egyes esetekben ez a két állapot megegyezik. Ezt többnyire olyankor jelöljük, ha ki szeretnénk hangsúlyozni, hogy az adott esemény nem vált ki állapot változást, vagy ha az eseményhez tevékenység is párosul.

Az eseményeket jelző nyílon szerepelhetnek szögletes zárójelek közt különböző előfeltételek, is. Ezen őrzőfeltételek nem teljesülése esetén, hiába következik be az adott esemény, az állapotváltozás nem megy végbe. Ritkán előfordulhat az is, hogy egy nyílon nincs esemény, ezek az automatikus állapotátmenet változások.

A tevékenységek vagy más néven akciók az eseménytől eltérően nem pillanatnyi történések, hanem időben elhúzódó műveletek. Amennyiben eseményhez tartozik tevékenység azt az esemény nyilán a neve után per '/' jellel elválasztva írjuk.

Belső tevékenységnek nevezzük azokat a tevékenységeket, melyek nem esemény hatására, hanem egy állapot közben hajtódnak végre. Ha nem tudunk az adott állapotnak nevet adni, gyakran használjuk a „do / tevékenység” mintát. Így az állapot neve maga a fő művelet, amit a rendszer az adott állapotban végez. Az „entry / tevékenység” jelöli az állapotba váltás közben végbemenő tevékenységet, az „exit / tevékenység” pedig az állapotból való kilépés közben végrehajtandó műveletet szemlélteti.

Egyes esetekben egy-egy állapotot kifejthetünk beágyazott állapot diagrammal, így az adott állapotban, több részállapotban is lehet a rendszer, ezt nevezik az állapotok aggregációjának is.

### Csomagdiagram

Az UML modellünkben szereplő különböző összetartozó elemek és funkcionalitások együtt kezelésére alkalmasak a csomagok. Ezek a csomagok magas szinten használati esetek csoportosítására hivatottak, de amennyiben osztályok csoportosítására használjuk ezeket, akkor a későbbi megvalósítás során tényleges névtérként vagy csomagként jelenhetnek meg, amennyiben erre az adott programozási környezet módot ad. A csomag a diagramon egy áttekintést nyújt a befoglalt elemeiről. A csomagok között leggyakrabban használt kapcsolati típus a függőség kapcsolat, mely azt fejezi ki, hogy az egyik csomag működéséhez felhasználja a másik csomagot, tehát függ tőle. Ezen a diagramon is értelmezett az úszósávok használata, amit többnyire a csomagok közti rétegződés (angolul layer) szemléltetésére használunk.

## Vegyes ki

Az elemzés során egyes módszerek azt ajánlják, hogy a követelményeket származásuk szerint három csoportba, azaz három nézőpontba soroljuk. Az interaktor nézőpontba a renszerrel közvetlen kapcsolatban álló, míg a közvetett nézőpontba a rendszerrel közvetlenül nem kapcsolatban álló kulcsfiguráktól erednek. A harmadik nézőpont pedig a rendszer felhasználási környezetből származó követelményeket magában foglaló szakterületi nézőpont.

A folyamat során érdemes a magas szintű felhasználói és nem funkcionális követelményeket megfogalmazni, melyeket a finomítás során rendszerkövetelményekre lehet tördelni.

### Célkitűzések

Érdemes a project megkezdése elején egy rövid áttekintést adni, a készítendő szoftver rendszer alapvető céljairól, betöltendő szerepéről. Ezek sokkal inkább alapvető iránymutatásként szolgálnak, mint tényleges követelményként. Ezek lehetnek általános víziók a szoftver hasznosságáról, főbb feladatairól, vázolhatja a főbb használati szituációkat, és környezetet. Ez a legáltalánosabb vázlat, amit a rendszerről készítünk. Érdemes lehet itt megemlíteni, hogy a felhasználók munkáját a termékünk miképpen fogja megkönnyíteni, milyen fontosabb értékeket hordoz a megrendelőknek.

### Rendszer

Rendszer alatt olyan egymáshoz kapcsolódó komponensek, működési egységelemek, halmazát értjük, melyek egy közös cél érdekében dolgoznak együtt. Ezek a rendszerkomponensek munkájuk elérése érdekében, kommunikálnak a többi komponenssel és felhasználhatják azokat. A rendszer tartalmazhat szoftveres, mechanikus, és elektronikus egységeket is. A rendszer célja elérése közben emberi beavatkozást is igényelhet, de lehet teljesen automatikus is.

### Nemfunkcionális követelmények

Az egész rendszerre vonatkozó tulajdonságok. Olyan eredő rendszertulajdonságokra vonatkozhat, mint a megbízhatóság, a válaszidő vagy a tárfoglalás. Lehet megszorítás egyes használt technológiákra, eljárásokra, vagy szabványokra vonatkozólag. Nem csak a teljes rendszer, de akár annak egy-egy összetevőjére is vonatkozhatnak. A míg, egy funkcionális követelmény nem megfelelő támogatásával a rendszer egyes részei megkerülendővé válhatnak, addig a nemfunkcionális követelmények nem teljesítése akár a teljes rendszert is használhatatlanná teheti. A nemfunkcionális nem csupán magára a rendszerre vonatkozhat, de magára a kifejlesztésének folyamatára is vonatkozhatnak. Megszabhatja tehát a kifejlesztés közben használt metodikát, különböző minőségszabványok betartását írhatja elő, vagy akár megszabhatja a fejlesztés alatt használatos CASE eszközök sorát.

Az ide sorolható változatos követelmények, különböző forrásokból is származhatnak, mint például a szoftver termék kívánt jellemzőiből, a fejlesztő vállalattól, vagy valamilyen külső féltől, mely többnyire a megrendelő, de lehet akár valamilyen előírás is.

Lehetséges, hogy egy adott nemfunkcionális követelmény megléte, szükségessé teszi más, funkcionális követelmények felvételét. Például egy információbiztonsággal kapcsolatos követelmény, megkövetelheti különböző authentikációs és authorizációs funkciók felvételét.

A nemfunkcionális követelmények egyik problematikája, hogy a megrendelők gyakran csak általános célokként fogalmazzák meg. Ilyen nehezen mérhető jelzőket használnak, mint gyors, kisméretű, egyszerűen kezelhető, hordozható, megbízható, stb. Ezek helyett célszerű a követelményekben különböző objektív metrikákat használni. A gyors, jelző kifejezhető inkább a másodpercenkénti tranzakciók számával, egy eseményre adott átlagos válaszidővel, vagy akár a képernyő frissítési sebességével. A méret leírható elfoglalt tárterület alapján, mint például nem lehet nagyobb 64 kilobájtnál, vagy a maximális dinamikus memóriafogyasztással. A könnyű használatot leírhatjuk a betartott tervezési ajánlásokkal, a képzési idővel, vagy a segédképernyők számával. A megbízhatóságot a rendelkezésre állás, a hibák közt eltelt átlagos idő, és a hiba bekövetkeztének valószínűsége írhatja le precízen. A hordozhatóságot jellemezheti például a célrendszertől független utasítások aránya.

## A szoftveréletciklus modellek és a követelmények kapcsolata

A szoftverfejlesztési módszertanok olyan módszerek, melyek meghatározzák a fejlesztés menetét és lépéseit. Keretet biztosítanak ahhoz, hogy milyen sorrendben végezzük az egyes a rendszer kifejezését elősegítő lépéseket. A módszertanok tartalmazhatnak különböző eszközöket a modellezésre, jelölési konvenciókat és ajánlásokat. Tanácsokat, esetlegesen betartandó szabványokat, tartalmazhatnak specifikációra, a tervezésre, illetve a fejlesztésre vonatkozólag.

### A hagyományos megközelítések problémái

Az olyan hagyományos módszertanokban, mint a vízesés modell, a fejlesztési lépések szigorú egymásutániságban követik egymást. Ezek a lépések a következők: követelményelemzés és meghatározás, a rendszer- és szoftvertervezés implementáció és egységteszt, az integráció és rendszerteszt, és végül egy működtetés és karbantartás, lépés. A módszer egyik nagy hibája abban áll, hogy ebben a merev rendszerben, gondot okoz, a követelmények módosulása, ugyanis ilyenkor esetleg az összes eddigi lépéseket újra végre kell hajtani. A követelmények minősége, és teljessége, kritikus, hisz a többi fázis ezekre építkezik, és a végső termék minősége is főként ezektől függ. A gyakorlatban számtalanszor előfordult, hogy a követelmények elemzése annyi időt és energiát emésztett fel, hogy az a teljes project bénulásához vezetett. Ezt a jelenséget hívja a szakirodalom elemzési bénultságnak (Analysis paralysis).

Egy másik probléma, hogy későn vesszük észre, ha esetleg nem pont azt a terméket készítjük, amit a felhasználó kíván, tehát későn következik be egy átfogó validáció, és későn látja működés közben a rendszert. Így a project kockázata döntő mértékben a fejlesztés végére csoportosul.

Az ilyen módszertanok, csak lassan változó követelményekkel rendelkező, teljesen leírható területekre szánt szoftvereknél alkalmazható hatékonyan.

### A manapság gyakori módszertanok és a követelmények kapcsolata.

A manapság legelterjedtebb életciklus modellek mind iteratív jellegű, ciklikus folyamatok. Ezekre általában igaz, hogy egyszerre a rendszer egy részét veszik, és ezen végigviszik, a fejlesztés lehetőleg összes lépését, majd az elkészült produktumot a felhasználónak meg is mutatják véleményezésre.

Az egyik ilyen metodika a RUP (Rational Unified Process), melyet az UML kifejlesztésén is fáradozó „három amigó”, James Rumbaugh, Grady Booch és Ivar Jacobson készítettek, a Rational cégnél. Ez a metodika egy iteratív és inkrementális modell. A szoftvert növekményekre bontja, ezeket az inkremenseket, a felhasználó szemszögéből fontossági sorrendbe állítja, és ezeken a részeken viszi végig a fejlesztési fázisokat. Így biztosított az, hogy a fontosabb rendszerkomponensek előbb elkészüljenek, ezeket a felhasználó, akár a teljes rendszer elkészülte előtt, igénybe is veheti. Ez a metodika ajánlja a teljes rendszer, különböző részletességű vizuális modelljeinek elkészítését. A módszer az üzleti folyamatok, illetve a követelmények modellezésére a használati eseteket ajánlja. És ezeket a modelleket fel is használja a teljes fejlesztési folyamat során. Fontosnak tartja a megrendelő követelményeinek explicit dokumentálását, és azok folyamatos karbantartását.

A napjainkban divatos Agilis elveket követő modellek is igyekeznek a rendszert kisebb a felhasználó számára értékeket hordozó egységekre bontani, és ezeket súlyozni, és fontosságuk szerinti sorrendben elkészíteni. Ezek az értékek egyenként áramolnak át a fejlesztés összes fázisán. A kezdeti célok és magas szintű üzleti modellezés és követelmény leírások itt is szerepet kapnak, de a végső követelmények a termékkel együtt fejlődnek. Az agilis szemléletmódot követő Scrum metodika, vagy az egyre inkább tért nyerő Lean megközelítés is a fejlesztés természetes velejárójaként tekint a követelmények változására, ugyanis sokszor a fejlesztés közben jobb képet kaphatunk a rendszerről, nagyobb tudás birtokában vagyunk. Emellett a világunk is folytonos változás állapotában van, így nem kérdés, hogy egy szoftver követelményei is megváltozhatnak. Az agilis projectekben inkább a felhasználó érdekeit igyekeznek szem előtt tartani, minthogy a követelmények szövegeihez próbálnának csökönyösen ragaszkodni, de ez nem jelenti azt, hogy az ilyen szemlélet szerint készülő szoftvereknek ne lenne specifikációja, sőt a követelmények modellezése itt is bevett szokás, de a formális diagramok helyett előszeretettel választják a formába öntött szövegek használatát.

Azzal, hogy a ciklikus modellek, a szoftvert több szakaszban adják át a megrendelőnek, a project kockázatát, a hagyományos modelleknél jobban képesek időben elosztani, és a project összkockázatát is sikeresen csökkentik. Mindemellett a gyakoribb kommunikációval és átadásokkal a követelmények esetleges félreértéséből adódó kockázatot is eredményesen csökkentik. Emellett viszont megnövelik a követemének karbantartásának és követésének a fontosságát. A nagyobb és komplexebb szoftverfejlesztési projecteknél elengedhetetlen, hogy a követelmények kezelését megfelelően támogató szoftvereszközöket alkalmazzanak.

## A CASE eszközök

A CASE (Computer-Aided Software Engineering) rövidítés számítógéppel támogatott szoftvertervezést jelent. Ellenben ezen eszközök jelenleg koránt sem csak a tervezés folyamatát támogatják.

A CASE rövidítés (Computer-Aided Software Engineering) számítógéppel támogatott szoftver tervezést jelent, de ezen eszközök jelenleg koránt sem csak a tervezés folyamatát támogatják.

A CASE eszközök olyan szoftverek vagy szoftver rendszerek, melyek a szoftverfejlesztés különböző fázisait és/vagy ezek közti kapcsolat megteremtését segítik elő, lehetőleg minél több automatikus eszköz és eljárás segítségével. A CASE eszközök többnyire egy fejlesztési metodikához illeszkednek, és ennek a szoftverfolyamatnak a tevékenységeit, vagy azok egy részét támogatják. Persze léteznek általánosan használható CASE eszközök is, melyesk nem kötődnek szorosan egy adott metodikához. A szoftverfolyamat tevékenységei közül támogatják például a követelményelemzést, a követelmények változásainak kezelését, a rendszermodellezést és a modellek esetleges ellenőrzését, a tervezést, a fejlesztés nyomkövetését, és akár a tesztelést. CASE eszközök segíthetik a követelmény specifikációs, és a szoftvertervezésnél használt modellek és diagramok előállítását. Egyes CASE eszközök a tervezés elemeiről adatszótárakat tartalmaznak, amik leírják az adott elemeket és azok kapcsolatait. Segíthetik a felhasználói interfész vázlatszerű, vagy teljes értékű elkészítését.

A CASE eszközök, a rutinfeladatok automatizálásával képesek a szoftver minőségén javítani, illetve a fejlesztés sebességét fokozni.

Bár nagymértékben megkönnyítik a szoftverek kifejlesztését, a CASE eszközök nem voltak képesek oly mértékben segíteni a munkát, mint amennyire azt a 1980-as, '90-es években prognosztizálták. Ennek fő okai közül az egyik, hogy a szoftverek kifejlesztése egy egyedi, és kreatív folyamat, melyet, akár mesterséges intelligencia bevonásával is, csak nehezen, vagy egyáltalán nem lehet teljesen automatizálni. A másik jelentős probléma, hogy egyes eszközök nem képesek elégségesen megtámogatni a csoportos munkavégzést, mely a nagy és komplex feladatoknál elengedhetetlen. Szerintem törekedni kell továbbá ezen CASE eszközök minél nagyobb fokú integráltságának, illetve együttműködésének biztosítására. Hisz hiába van egy olyan eszközünk, ami az adott részfolyamatot tökéletesen segíti, de az adott rendszer feltöltése a szükséges adatokkal több időt elvesz, mintha egy olyan rendszert használnánk, amely esetleg nem olyan hatékony céleszköz, de a környező rendszerekkel való kommunikációja révén a feladatot mégis gyorsabban képes ellátni, és a fejlesztési folyamatban az értékek áramlását, kevésbé gátolja.