Óbudai Egyetem

Neumann János Informatikai Kar

Alkalmazott Informatikai Intézet

SZAKDOLGOZAT

FELADATLAP

Hallgató neve: **Lovas István**

Törzskönyvi száma: T/002145/FI12904/N

A dolgozat címe:

**Szoftver követelmények feltárását, analizálását és modellezést támogató intelligens eszköz**

**Intelligent Tool to Support Software Requirements Elicitation, Analysis and Modelling**

Intézményi konzulens: Dr. Tick József

Külső konzulens:

Beadási határidő: 2015. január 7.

A záróvizsga tárgyai: Számítógép architektúrák

Szoftverfejlesztés

**A feladat**

Készítsen szakdolgozatot, melyben bemutatja a szoftver követelménytervezés, illetve a követelmények modellezésének általános megközelítéseit. A szakirodalom alapján ismertesse az alkalmazott elveket, megoldásokat és azon eszközöket, melyek ezt a tevékenységet támogatják. Tervezzen meg, és valósítson meg egy olyan elosztott szoftver-eszközt, mely segíti az akár csoportos követelmény feltárást, követelmény analízist, a követelmények modellezését, összeköti a specifikáció szöveges részeit, a modellreprezentációk különböző absztrakcióival. Az eszköz könnyítse meg az ezek közti navigációt, illetve nyújtson intelligens módon segítséget a modellek elkészítésében, a szövegekből kinyert információk alapján. Az eszköz fejlesztése során alkalmazza a szoftvertechnológia modern megoldásait.

**A dolgozatnak tartalmaznia kell**:

* A feladat részletes leírását.
* Követelménytervezés elterjedt megközelítéseinek bemutatását a szakirodalom alapján.
* A kapcsolódó szövegbányászati eljárások szakirodalom segítségével történő bemutatását.
* A rendszer tervét, UML alkalmazásával.
* Az eszköz megvalósításának leírását
* A rendszer tesztelését, és annak eredményeit.
* Az eszköz továbbfejleszthetőségi lehetőségeit.

Ph.

.....……………….

Dr. Galántai Aurél

mb. intézetigazgató

A szakdolgozat elévülésének határideje: **2017. január 7.**

(OE TVSz 32.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

|  |  |
| --- | --- |
| ……………….. | ..……………………. |
| külső konzulens | intézményi konzulens |

# Tartalomjegyzék

[**Szoftver követelmények feltárását, analizálását és modellezést támogató intelligens eszköz** 1](#_Toc405716218)

[1 Tartalomjegyzék 3](#_Toc405716219)

[2 Ábrajegyzék 8](#_Toc405716220)

[3 Célkitűzések 9](#_Toc405716221)

[4 Irodalomkutatás 10](#_Toc405716222)

[4.1 Bevezetés 10](#_Toc405716223)

[4.2 A Követelménytervezés alapfogalmai 10](#_Toc405716224)

[4.2.1 Célkitűzések 10](#_Toc405716225)

[4.2.2 Szoftverkövetelmény 10](#_Toc405716226)

[4.2.3 Követelmények csoportosításai 11](#_Toc405716227)

[4.2.4 Felhasználói követelmények 11](#_Toc405716228)

[4.2.5 Rendszerkövetelmények 11](#_Toc405716229)

[4.2.6 Rendszer 12](#_Toc405716230)

[4.2.7 Funkcionális követelmények 12](#_Toc405716231)

[4.2.8 Nemfunkcionális követelmények 12](#_Toc405716232)

[4.2.9 Szakterületi követelmény 13](#_Toc405716233)

[4.2.10 Szükséges és javasolt követelmények 14](#_Toc405716234)

[4.2.11 Kulcsfigura 14](#_Toc405716235)

[4.2.12 Verifikáció és Validáció 14](#_Toc405716236)

[4.2.13 Követelmények kezelése 14](#_Toc405716237)

[4.2.14 Fogalomszótár 15](#_Toc405716238)

[4.3 A követelménytervezés folyamata 15](#_Toc405716239)

[4.3.1 Megvalósíthatósági tanulmány elkészítése 16](#_Toc405716240)

[4.3.2 Követelmények feltárása és elemzése 16](#_Toc405716241)

[4.3.3 Követelmény specifikációs folyamat 18](#_Toc405716242)

[4.3.4 Szoftverkövetelmények dokumentuma 18](#_Toc405716243)

[4.3.5 Követelmény validáció 19](#_Toc405716244)

[4.4 A követelmények fontossága 19](#_Toc405716245)

[4.5 A követelmény feltárás, és kezelés problematikái 20](#_Toc405716246)

[4.5.1 A természetes nyelv problémái 20](#_Toc405716247)

[4.5.2 A természetes nyelv problémáit kiküszöbölendő megfontolások 20](#_Toc405716248)

[4.6 A szoftveréletciklus modellek és a követelmények kapcsolata 21](#_Toc405716249)

[4.6.1 A hagyományos megközelítések problémái 21](#_Toc405716250)

[4.6.2 A manapság gyakori módszertanok és a követelmények kapcsolata. 22](#_Toc405716251)

[4.7 Követelménymodellezés UML segítségével 24](#_Toc405716252)

[4.7.1 Bevezető gondolatok 24](#_Toc405716253)

[4.7.2 Az UML bemutatása 24](#_Toc405716254)

[4.8 Az üzlet és a követelmények modellezése UML segítségével 27](#_Toc405716255)

[4.9 A követelmények modellezése közben használt diagram eszközök 28](#_Toc405716256)

[4.9.1 A használati-eset diagram 28](#_Toc405716257)

[4.9.2 A tevékenység diagram 30](#_Toc405716258)

[4.9.3 Sztereotípiákkal ellátott elemzési osztálydiagram 31](#_Toc405716259)

[4.9.4 Szekvencia diagram 32](#_Toc405716260)

[4.9.5 Együttműködési diagram 33](#_Toc405716261)

[4.9.6 Állapotdiagram 34](#_Toc405716262)

[4.9.7 Csomagdiagram 35](#_Toc405716263)

[4.10 Az agilis követelménykezelés és modellezés 35](#_Toc405716264)

[4.10.1 User Stories: A használati esetek agilis megközelítése 36](#_Toc405716265)

[4.10.2 A TDD, DDD, BDD hármas értelmezése 37](#_Toc405716266)

[4.10.3 Gherkin nyelven leírt és „futtatható” követelmények. 38](#_Toc405716267)

[4.11 Követelmény formátum sablonok 40](#_Toc405716268)

[4.11.1 Felhasználói követelmény sablon 40](#_Toc405716269)

[4.11.2 Rendszerkövetelmény sablon 40](#_Toc405716270)

[4.11.3 Formális használati eset szcenárió sablon 40](#_Toc405716271)

[4.11.4 User Story sablon 41](#_Toc405716272)

[4.11.5 Gherkin sablon 41](#_Toc405716273)

[4.12 A CASE eszközök 42](#_Toc405716274)

[4.13 A szövegbányászatról általánosságában 43](#_Toc405716275)

[4.14 Szövegbányászati előfeldolgozás 44](#_Toc405716276)

[4.15 Szövegbányászati modellalkotás 44](#_Toc405716277)

[4.15.1 Az ismertebb dokumentum reprezentációs modellek bemutatása 44](#_Toc405716278)

[4.15.2 A vektortér modell 44](#_Toc405716279)

[4.15.3 A szó-dokumentum mátrix jellemző súlyozási sémái 45](#_Toc405716280)

[4.15.4 Módok a szöveg felbontására, illetve a szótár felépítésére 46](#_Toc405716281)

[4.15.5 A strukturális szegmentálás 47](#_Toc405716282)

[4.15.6 Mondatokra bontás 47](#_Toc405716283)

[4.15.7 Tokenizálás 47](#_Toc405716284)

[4.15.8 Stoppszószűrés 48](#_Toc405716285)

[4.15.9 Lemmatizálás és szótövezés 49](#_Toc405716286)

[4.16 Információkinyerés 50](#_Toc405716287)

[4.16.1 Az információkinyerés összehasonlítása más főbb feladattípusokkal 50](#_Toc405716288)

[4.17 Az információkinyerés fontosabb részfeladatai 51](#_Toc405716289)

[4.17.1 A névelem felismerés 51](#_Toc405716290)

[4.17.2 A szereplők azonosítása 51](#_Toc405716291)

[4.17.3 A kereszthivatkozások azonosítása 52](#_Toc405716292)

[4.17.4 A szereplők közti relációk azonosítása 52](#_Toc405716293)

[4.17.5 Események felfedése és illesztése részfeladat 52](#_Toc405716294)

[4.18 Az információkinyerés során előforduló főbb nyelvészeti problémák 52](#_Toc405716295)

[4.19 A szabály és a statisztika alapú információkinyerési megközelítések összevetése 53](#_Toc405716296)

[4.19.1 A statisztikai megközelítések által használt elterjedt reprezentációs modellek 54](#_Toc405716297)

[4.20 További szövegelemzési megfontolások 54](#_Toc405716298)

[4.21 Az irodalomkutatás alapján leszűrt tanulságok 55](#_Toc405716299)

[4.21.1 Egy követelményelemzést támogató eszköz főbb tulajdonságai. 55](#_Toc405716300)

[4.21.2 Projektszótárak és szövegszínezés 56](#_Toc405716301)

[4.21.3 Az információ kinyerés lehetséges megfontolásai az adott feladatkörben 57](#_Toc405716302)

[4.21.4 Mik lehetnek egy modern CASE eszköz főbb tulajdonságai. 57](#_Toc405716303)

[4.21.5 A szöveg-kiegészítésnél alkalmazható megfontolások 57](#_Toc405716304)

[4.21.6 A modell elkészítését segítő módszerek. 59](#_Toc405716305)

[5 Már létező hasonló jellegű programok vizsgálata 59](#_Toc405716306)

[5.1 Sparx Enterprise Architect 60](#_Toc405716307)

[5.1.1 Követelményfolyamat támogatása 60](#_Toc405716308)

[5.1.2 A modellezés támogatása 60](#_Toc405716309)

[5.1.3 Csoportos munka támogatása 60](#_Toc405716310)

[5.2 Visual Paradigm 60](#_Toc405716311)

[5.2.1 Követelményfolyamat támogatása 61](#_Toc405716312)

[5.2.2 A modellezés támogatása 61](#_Toc405716313)

[5.2.3 Csoportos munka támogatása 61](#_Toc405716314)

[5.3 Értékelés és az elkészítendő rendszer a két létező rendszer tükrében 61](#_Toc405716315)

[6 A rendszerrel szemben támasztott követelmények 62](#_Toc405716316)

[6.1 Vázlatos felhasználói követelmények, illetve igények. 62](#_Toc405716317)

[6.2 A rendszerben megjelenő felhasználói szerepkörök 64](#_Toc405716318)

[6.2.1 A rendszer felhasználóinak használati eset nézete 64](#_Toc405716319)

[6.2.2 A rendszerben megjelenő, előre definiált felhasználói szerepkörök jellemzése 64](#_Toc405716320)

[6.3 Funkcionális követelmények 66](#_Toc405716321)

[6.3.1 A rendszer főbb felhasználói funkcióinak áttekintő használati eset nézete 66](#_Toc405716322)

[6.4 Nem funkcionális követelmények 67](#_Toc405716323)

[7 A rendszer kialakítása során felhasználható technológiák és jellemzésük 68](#_Toc405716324)

[7.1 Szerver oldali technológiák 68](#_Toc405716325)

[7.1.1 ASP.NET 68](#_Toc405716326)

[7.1.2 Java 68](#_Toc405716327)

[7.2 A kliens oldali megjelenítés kezelése 68](#_Toc405716328)

[7.2.1 HTML5 68](#_Toc405716329)

[7.2.2 CSS 68](#_Toc405716330)

[7.3 A kliens oldali dinamika kezelése 68](#_Toc405716331)

[7.3.1 JavaScript, és a TypeScript 68](#_Toc405716332)

[7.4 Kommunikációs technológiák 68](#_Toc405716333)

[7.5 Szövegbányászati csomagok 68](#_Toc405716334)

[8 Követelményanalízis 69](#_Toc405716335)

[8.1 A rendszer statikus modelljének elemzése 69](#_Toc405716336)

[8.1.1 Szerkezet áttekintő nézet 69](#_Toc405716337)

[8.1.2 Szerkezeti nézet 70](#_Toc405716338)

[8.2 A rendszer dinamikájának és folyamatainak elemzése 70](#_Toc405716339)

[8.3 Képernyő vázlatok 71](#_Toc405716340)

[9 Rendszerterv 71](#_Toc405716341)

[9.1 Képernyő tervek 71](#_Toc405716342)

[9.2 Csomagáttekintő nézet 71](#_Toc405716343)

[9.3 Részletes komponens leírások 72](#_Toc405716344)

[9.4 Részletes dinamikus nézetek 78](#_Toc405716345)

[9.5 Telepítési nézet és rendszerkörnyezet 82](#_Toc405716346)

[10 Az implementáció részletei 83](#_Toc405716347)

[11 Tesztelés 89](#_Toc405716348)

[11.1 Statikus tesztelés 89](#_Toc405716349)

[11.2 Dinamikus tesztelés 90](#_Toc405716350)

[11.2.1 Egység teszt 90](#_Toc405716351)

[11.2.2 Komponens teszt 91](#_Toc405716352)

[11.2.3 Integrációs teszt 92](#_Toc405716353)

[11.2.4 Rendszer teszt 93](#_Toc405716354)

[11.2.5 Átadási teszt 94](#_Toc405716355)

[12 Továbbfejlesztési lehetőségek 95](#_Toc405716356)

[13 A szakdolgozat tartalmi összefoglalója 96](#_Toc405716357)

[14 Irodalomjegyzék 98](#_Toc405716358)

[15 Mellékletek 99](#_Toc405716359)

# Ábrajegyzék

# Célkitűzések

Célunk egy olyan eszköz kifejlesztése, ami képes segíteni a szoftverfejlesztés egyes magas absztrakciós tevékenységeit. Gondolok itt főképp felhasználók által elkészítendő szoftver-rendszer követelményeinek specifikációjával, analízisével, illetve modellezésével kapcsolatos teendőkre. A készítendő rendszernek célja továbbá, hogy támogassa a csoportos munkát, lehetőleg úgy, hogy a különböző felhasználók valós időben láthassák egymás munkáit és közben képesek legyenek kommunikálni is egymással, a rendszert felhasználva. Végül, de nem utolsó sorban a rendszer legyen a lehető legkülönbözőbb környezetekből, és platformokról elérhető, és használható.

# Irodalomkutatás

## Bevezetés

Az elkövetkezendő szakaszokban, a feldolgozott irodalmak alapján bemutatom a szoftverkövetelményeket, a követelménytervezés folyamatát, kitérek a követelmények modellezésére, a szakirány központú szemléletekre, és az agilis követelménykezelésre. Majd röviden írok az egyes szoftver életciklus modellekről, illetve a CASE eszközökről. Ezután a szövegbányászat alapvető koncepcióit mutatom be, és kitérek néhány alternatív, egyszerű, a szoftverkövetelményekkel kapcsolatos információkinyerési megközelítésre.

## A Követelménytervezés alapfogalmai

Ebben a szakaszban igyekszem vázolni, a követelményekkel kapcsolatos összes fogalmat. Egyes fontosabb fogalmakra a későbbiekben külön fejezetben részletesebben kitérek. Nem foglalkozok ellenben a kritikus rendszerek speciális igényeivel, illetve a formális specifikációk mélyebb elemzésével, mert ezek a dolgozat hatókörén kívül esnek. A szakasz az Ion Somerwille könyve [1] alapján készült.

### 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.

### Szoftverkövetelmény

A rendszer követelményei lehetnek a rendszer funkcionalitására vonatkozó elvárások magas és-vagy alacsonyabb szintű, részletesebb megfogalmazása, illetve a rendszerrel szemben támasztott megszorítások. Megmondja, hogy a rendszer a megrendelő problémáit milyen szolgáltatások révén oldja meg, és megoldás közben milyen általános és probléma centrikus megkötésekre kell odafigyelni. Részletesség szerint két csoportját érdemes megkülönböztetni, a felhasználói-, illetve a rendszerkövetelményeket. Ezeket a csoportokat szoktuk vegyíteni is tehát léteznek a funkcionális felhasználói követelmények, a nem-funkcionális felhasználói követelmények, és a rendszerkövetelmények is lehetnek funkcionálisak, illetve nem-funkcionálisak is. Ezeket az eltérő követelményeket fontos lehet elkülöníteni egymástól, akár jelöléssel, akár úgy, hogy a követelmény dokumentáció külön alfejezeteit alkotják az imént vázolt csoportok, ellenben jó ötlet lehet, az összekapcsolódó, de különböző csoportba sorolt követelmények gyors összevetésére is módot adni, de legalább egy hivatkozást elhelyezni, a kapcsolódó követelményre.

### Követelmények csoportosításai

Követelmények csoportosítása történhet a megfogalmazásuk mélysége szerint. Ugyanis a gyakorlatban fontos szétválasztani a követelmények azon szintjét, mely a megrendelőkkel, illetve a felhasználókkal történő egyeztetést segítik, ezek a felhasználói követelmények, és azt a szintet, melyben a rendszer követelményeit részletezzük, ezeket nevezzük rendszerkövetelményeknek. A követelmények egy másik csoportosítása történhet az alapján, hogy az adott követelmény egy a rendszertől elvárt szolgáltatást részletez, azaz funkcionális követelmény, vagy egy a rendszertől, esetleg annak egyes szolgáltatásaitól elvárt tulajdonságot ír le, tehát nem-funkcionális követelmény.

### Felhasználói követelmények

Absztrakt módon leírja a rendszertől elvárt szolgáltatásokat, külső viselkedését, és azok működési megszorításait. Közérthető, természetes nyelvű leírás, melyet a rendszer felhasználójának, a megrendelőnek, és ezek megbízottjainak szánnak. Esetenként a könnyebb érthetőség kedvéért kiegészítik magas szintű, vázlatos használati eset diagramokkal.

A természetes nyelven megfogalmazott követelmények, az olyan egyértelmű előnyeik mellett, mint az egyszerűségük, olyan problémákat is felvetnek, mint az egyértelműség nehéz biztosítása, ezekkel: A követelmény feltárás, és kezelés problematikái című részben bővebben foglalkozom.

### Rendszerkövetelmények

A rendszer szolgáltatásait, funkcióit, működési feltételeit és megszorításait részletezi. A felhasználói követelményeket fejtik ki bővebben. Érdemes megjegyezni, hogy a rendszerkövetelmények dokumentumát szokták specifikációnak is hívni.

Lényeges a specifikáció pontossága, hisz meghatározza, hogy mit kell implementálni, de ezen a szinten már kivédhetetlenül megjelenhet a hogyan is, emellett azért is fontos a teljes körű és precíz megfogalmazás mert a rendszerspecifikáció gyakran része a szerződésnek.

A rendszerspecifikációból lehetetlen kizárni minden a tervezéshez köthető információt, mert gyakran a követelményspecifikáció strukturálásához segítséget nyújt egy kiindulási architektúra elkészítése. A gyakorlatban emellett gyakran szükséges, hogy együtt működjünk más rendszerekkel, és ez bekorlátolja a tervezést és ezek a korlátok szükségszerűen új követelményekként is megjelenhetnek. Esetenként egy-egy nemfunkcionális követelmény is előírhat egy konkrét tervezési megközelítést, vagy egy konkrét fejlesztési platformot, így ilyenkor is megjelennek tervezési részletek, a rendszerkövetelmények szintjén.

A rendszerkövetelmények tehát megalapozzák a tervezést, illetve az implementációt, így a szoftver megalkotásában résztvevők a legfőbb olvasói. Ezért míg a felhasználói követelményeknél elengedhetetlen a természetes nyelv, és a külön képzettség nélküli érthetőség, addig a rendszerkövetelmények szintjén, az egyszerű formális szövegek mellett speciális jelöléseket és modelleket is alkalmazhatunk. Ilyenek lehetnek a stilizált, formázott és strukturált természetes nyelv, vagy a követelmények grafikus modelljei, mint a részletezett használati esetek vagy akár a matematikai formális nyelvek.

### 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.

### Funkcionális követelmények

Leírja a rendszertől elvárt szolgáltatásokat. Az iménti fejezetekből világosan látható, hogy két szintje lehetséges, a funkcionális felhasználói követelmények magas szintű állítások, amik megfogalmazzák a rendszertől elvárt funkcionális viselkedést, míg a funkcionális rendszerkövetelmények, e funkcionalitás részletezései.

### 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.

### Szakterületi követelmény

A szakterületi, vagy más néven környezeti követelmények, a rendszer alkalmazási területéről származnak. Ezek a követelmények nem éppen felhasználói igények, inkább a szoftver alkalmazásának szakterületéből adódó funkcionalitások, vagy megszorítások.

Ezek teljes értékű funkcionális, vagy nemfunkcionális követelmények, de mind a nyelvezetük, mind fogalomrendszerük az adott szakterülethez igazodik. Ha funkcionális követelmény, akkor előírhatja, hogy az adott feladatot hogyan kell végrehajtani, hogy az helyes legyen, illetve illeszkedjen az adott szakterület bevett gyakorlatához. Lehet akár egy szabvány, de lehet akár egy képlet is, amivel az adott szakterületen számolnak.

Ezen követelmények nagy jelentőségűek, míg az adott szakterület képviselőinek egyértelműek addig a rendszer tervezőinek némiképp idegenek lehetnek a nyelvezetük idegensége és a mögöttes implicit háttértudás hiányában.

### Szükséges és javasolt követelmények

A javaslat, nem mindig követelmény, lehet, hogy csupán egy igény a rendszerre vonatkozólag. Esetleg egy teoretikus elképzelés, ami jelenleg nem fontos, vagy kritikus és nem döntő fontosságú a rendszer szempontjából. Érdemes lehet ezeket is rögzíteni, hisz nyerhetünk belőlük újabb jó ötleteket vagy az is lehet, hogy egy átpriorizálás közben felértékelődik, és szükségessé nyilváníthatják.

A javasolt követelmények nyelvezetében érdemes, ilyen kifejezéseket használni, mint: „jó lenne ha”, vagy „érdemes lenne”, „javasolt lenne”, „javallott” stb.

Míg a szükséges követelmények a „kell”, a „szükséges”, a „fontos”, a „kívánt” az „elvárt”, és hasonló szavak használata különíti el az előző csoporttól.

### Kulcsfigura

A kulcsfigurának nevezzük a rendszerrel kapcsolatba kerülő végfelhasználókat, és az összes olyan egyént, akire a szoftverrendszer bármilyen hatást gyakorol. Ez a hatás lehetnek például a szoftver beüzemelése közben előforduló fennakadások hatásai, vagy akár a megrendelő cégen belüli személyek, az új rendszer megjelenésének következtében történő, politikai erőviszonyának megváltozása.

### Verifikáció és Validáció

Ezt a két lépést együtt szokták V&V-nek is rövidíteni. Míg a verifikáció azt ellenőrzi, hogy a szoftvert jól, azaz a specifikációknak megfelelően készítettük-e el, addig a validáció azt ellenőrzi, hogy tényleg a megfelelő, a felhasználó által óhajtott, értékes terméket alkottuk meg.

### Követelmények kezelése

A követelmények kezelése napjaink gyorsan változó világában igen fontos, ugyanis a rendszerrel kapcsolatos elvárások az idő előrehaladtával folyamatosan változhatnak. Megváltozhatnak az adott funkcionalitást igénylő üzleti folyamat, egyes relatív minőséget befolyásoló elvárások, vagy akár maga a megrendelő vállalat is. Ezzel a változással úgy tarthatjuk a lépést, ha az alkalmazással szembeni követelményeket és ezután magát a szoftverrendszert is a megváltozott érdekekhez szabjuk. A követelmények között vannak olyanok, amik nem, vagy csak igen lassan változnak, mint a megrendelő alapvető tevékenységével, vagy a szakterület alapjaival kapcsolatosak, ezeket hívjuk tartós követelményeknek, vannak ezen kívül gyakrabban módosuló, átmeneti követelmények is.

A követelmények kezelése gyakran igen sok erőforrást felemésztő és költséges feladat, ezért érdemes a követelménytervezés kezdeti szakaszában elhatározni a kívánt szintjét és megtervezni a menetét. A kezelés kisszámú követelmény esetén még elvégezhető kézi módszerekkel, de egy bizonyos mennyiség után elkerülhetetlen egy CASE eszköz használata.

A követelménykezelést megkönnyítendő érdemes a követelményekhez egy azokat a teljes rendszer szintjén azonosítani képes egyedi azonosítót rendelni. Az azonosítót felhasználhatjuk a rendszer további elemeivel illetve más dokumentumokkal való összekapcsolásra.

A követelmények nyomon követhetőségét elősegítheti, ha információt tárolunk a követelmények forrásáról, azaz azon kulcsfigurákról, akik az adott követelményt igényelték. Érdemes az egymástól függő követelményeket összekapcsolni, illetve jelölni, ha ugyanazon alrendszer egy másik követelménye módosult. A követelményeket továbbá érdemes összekapcsolni az az azokat megvalósító tervmodullal. Az ilyen kapcsolatok nagyban megkönnyítik a változás rendszerbeli kiterjedésének és a rendszertervre valamint az implementációra való kihatásának becslésére.

A változáskezelési folyamat olyan tevékenységeket tartalmaz, melyek segítségével a változás kiterjedését és költségét becsülhetjük. Ez a formális folyamat, segíti a változás rendezett és egységes módon történő kivitelezését. A folyamat első szakaszában elemezzük a változtatási szándék és probléma validitását, majd nyomon követési információk segítségével elemezzük a változtatás kihatásait, végső lépésként pedig végrehajtjuk először a követelményeken, majd a rendszerterven és végül az implementációban a változást.

### Fogalomszótár

A követelmények között, vagy a különböző modelleken megjelenhetnek olyan szavak, melyeket érdemes lehet definiálni. Ez több célt is szolgálhat, növeli a szövegek precizitását, megóvhat az egyes félreértésektől, kiküszöböli a kétértelműségeket. A szakterületi kifejezések leírása segítheti a fogalmak megértését, és ez által segítheti a hatékonyabb fejlesztést. A rendszerben használt rövidítéseket is érdemes lehet ide felvenni.

## A követelménytervezés folyamata

Ebben a folyamatban megértjük és definiáljuk a rendszer által biztosítandó szolgáltatásokat, illetve a fejlesztési és az üzemeltetési megszorításokat. A folyamat végeredménye a követelménydokumentum, mely rend szerint külön tárolja a felhasználói illetve a rendszerkövetelményeket.

A folyamatban elemezzük a rendszer fontosságát, majd felderítjük, elemezzük, dokumentáljuk és ellenőrizzük a szoftverkövetelményeket. Ennek megfelelően a folyamat négy nagy tevékenységre bontható. Ezek a részfolyamatok a megvalósíthatósági tanulmány elkészítése, a követelmények feltárása és elemzése, a követelmények validálása, illetve a követelmények kezelése és követése.

### Megvalósíthatósági tanulmány elkészítése

Megvizsgálja és becslést ad arról, hogy a rendszerrel kapcsolatos elvárások kielégíthetőek-e az adott szoftveres és hardveres technológiák segítségével. Eldönti, hogy a rendszer költséghatékony-e az adott üzleti szempontokat figyelembe véve, illetve hogy a költségvetési megszorítások mellett kivitelezhető-e. Lehetőség szerint minél inkább gyors, és olcsó folyamatnak kell lennie. A tanulmány elkészítése során kiszámításra kerül egy ROI (Return Of Investment) érték, mely megadja, hogy milyen mértékben és mikor térül meg az elkészítendő rendszer, és mekkora annak az üzleti haszna. A megvalósíthatósági tanulmány információt biztosít a rendszer elkészítésének költséghatékonyságáról.

A tanulmány elkészítésének végeztével döntés születik a fejlesztés folytatásáról. A megvalósíthatósági jelentés a részfolyamat kimenő dokumentuma, mely tartalmazza a lépésben összegyűjtött fontosabb információkat és döntéseket. Ez után a lépés után következhet a követelmények feltárása részfolyamat.

### Követelmények feltárása és elemzése

A folyamat során a potenciális felhasználókkal és megrendelőkkel történő megbeszélések, és egyeztetések során, illetve az esetleges már működő rendszerek és folyamatok megfigyelése által az elemzők feltérképezik és megértik a készítendő szoftver követelményeit. Ez a folyamat magában foglalhatja egyes rendszermodellek, illetve prototípusok elkészítését, melyek elősegítheti a követelmények pontosabb megértését.

A követelmény feltárást megnehezítheti, hogy a rendszerrel kapcsolatos, érintett kulcsfigurák pontos személye nem ismert, vagy azok nem ismerik, vagy nem képesek pontosan kifejezni azt, hogy mit várnak el a rendszertől. Továbbá az is előfordulhat, hogy a különböző kulcsfiguráknak eltérő vagy akár egymáséinak ellentmondó igényei vannak. A követelmények elemzése és priorizálása során figyelembe kell venni a rendszert befolyásoló üzleti és gazdasági környezetet.

A követelmény feltárási és elemzési folyamat felderítési-, osztályozási-, szervezési-, priorizálási- és dokumentálási lépései a folyamat során ciklikusan követik egymást. Így a már felderített követelmények folyamatosan fejlődnek, és új követelmények bukkanhatnak fel a folyamat során.

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.

A követelmények felderítése során segítségünkre lehetnek a különböző hasonló rendszerek megfigyelése, a rendszerrel kapcsolatba kerülő másik rendszerek megfigyelése, a kulcsfigurákkal folytatott interjúkból leszűrt tanulságok, prototípusok elkészítése és azoknak a kulcsfigurákkal történő elemzése.

Az interjúkat két típusba soroljuk, az egyik a zárt interjú, ahol előre meghatározott kérdéssorra válaszolnak az alanyok, és van a nyitott interjú, ahol az interjúztatók a kulcsfigurákkal szabadon bejárnak egy-egy problémakört. Az interjú módszere segíthet számos követelmény és igény felderítésében, és a rendszer későbbi felhasználóinak alaposabb megismerésében, de önmagában nem képes a teljes rendszer összes szükséges követelményét felfedni. Az interjúk során nehézséget okoz, hogy a kulcsfigura szakterületi szakzsargonját az interjúztató esetleg félreérti, illetve az interjúalanyok esetleg nem említenek meg általuk triviálisnak gondolt, de amúgy a rendszer építésénél kritikus fontosságú részleteket. A módszer további hátránya, hogy az interjú alanyok olykor hajlamosak a valóságos folyamat helyett azt elmagyarázni, ahogy a rendszerüknek a papírforma szerint működnie kéne.

A követelmény feltárás másik eszköze a forgatókönyvek vagy más szóval szcenáriók készítése. Mely megkönnyítheti a kulcsfigurákkal való precíz kommunikációt. Ugyanis azok ilyenkor a konkrét problémára, magára az üzleti folyamatra tudnak koncentrálni és azzal kapcsolatban véleményt formálni. A forgatókönyvek különböző eszközökkel és különböző formalitási szinteken készíthető. A teljesen informális szöveges leírással szemben célszerű különböző fél formális szcenárió sablonok segítségével, vázlatpontokba szedni az üzleti folyamatok forgatókönyv ágait. Az UML aktivitás diagramja is egy kiváló formális eszköz lehet a forgatókönyvek szemléletes modellezésére. A forgatókönyvek felderítésére egy módszer a használati esetek módszere, ahol a rendszer felhasználóinak szemszögéből modellezzük a rendszert. Egy másik napjainkban gyere nagyobb népszerűségnek örvendő módszer a követelmények felderítésére User Story-k használata és a szcenáriók Gherkin nyelvű leírása. A forgatókönyvek UML segítségével való-, illetve azok Agilis megközelítésű modellezésével a későbbiekben még részletesebben foglalkozok.

A követelmény feltárás egy további eszköze lehet az etnográfiai megfigyelés, mely során megvizsgáljuk a megrendelő társadalmi és szervezeti viszonyait. Az etnográfiai megfigyelést végző személy megvizsgálja továbbá a kulcsfigurák napi munkarutinját, az elkészítendő szoftverrendszert érintő valós munkafolyamatokat, az ügymenetet és a munkakörnyezetet.

### Követelmény specifikációs folyamat

Az elemzési tevékenység során összegyűjtött információk egységes dokumentummá történő szerkesztésének folyamata. Ez a szoftverspecifikáció megalkotásának folyamata, itt készítjük el és tartjuk karban a szoftverkövetelmények dokumentumát.

### Szoftverkövetelmények dokumentuma

A szoftverkövetelmények dokumentuma, amit szoktak szoftverkövetelmény specifikációnak is hívni, az a dokumentum, mely a követelmény specifikáció folyamata során jön létre. A specifikáció tehát a követelmények egy szabványos formába való szedésének és leírásának folyamata. Maga a dokumentum tartalmazza a felhasználói-, illetve a rendszer követelményeket. Szöveges dokumentum, így a megrendelő külön előképzettség nélkül is olvashatja, de a kötött szerkezete elősegíti a lehető legteljesebb információreprezentálást.

A funkcionális és a nemfunkcionális követelményeket célszerű megkülönböztetni a dokumentumon belül. Ez lehet elszeparálás, ilyenkor a külön elhelyezkedő követelmények között nehezebb megtalálni az esetleges összefüggéseket, viszont az egyes követelmények nem mosódnak egybe, és jól elkülöníthetőek a funkcionális, illetve a nemfunkcionális megfontolások. A megkülönböztetés történhet esetleg más vizuális jellemzőkkel is.

A dokumentum tartalmára vonatkozólag az IEEE/ANSI 830-1998-as szabvány nyújthat támpontot. Erre építkezve, ezt kibővítve Ian Sommerville a következő tartalmi részeket különbözteti meg:

Az első fejezet egy előszó, mely a dokumentum által megcélzott olvasókört íja le, és áttekintést nyújt a rendszer adott legfrissebb verziójával kapcsolatban. Majd következik az IEEE szabványban is ajánlott bevezetés szekció, mely a szoftverrendszer céljait, kifejlesztésének indokait, és áttekintést ad a tőle elvárt főbb funkciókról, és arról, hogy ez milyen módon illeszkedik, a megrendelő további üzleti illetve informatikai rendszereihez. Ajánlatos egy szekciót a dokumentumban használt szavak szójegyzékének szánni. Ezután következnek a felhasználói követelmények, a rendszer magas absztrakciós felépítése, majd a rendszerspecifikáció. Ezt követi egy a rendszermodelleket tartalmazó rész. Ezután egy a rendszer eddigi evolúcióját, az idő közben bekövetkezett fontosabb hardveres és szoftveres változásokat. A dokumentumot egy függelék és egy tárgymutató fejezet zárja.

### Követelmény validáció

A követelmények valószerűségét, konzisztenciáját és teljességét ellenőrző tevékenységet hívjuk validálásnak. Azért van szükség a követelmények validálására, hogy meggyőződjünk arról, hogy tényleg azt a rendszert készítjük-e el, amire a megrendelőnek szüksége van. A szoftver hibáinak javítási költsége annál alacsonyabb, minél előbb észrevesszük azt, tehát, ha a követelményekben van hiba, célszerű azt, ha lehet még a követelménytervezés folyamatában, a validálás során észrevenni. A követelményekben vétett hiba kihatással van az összes további fejlesztési lépésre és akár tévútra is vihetik az egész szoftverprojectet. A szoftverkövetelmények dokumentumának ellenőrzése közben számos különböző validálási lépést végrehajthatunk. A validitás-ellenőrzés során azt vizsgáljuk, hogy az összes kulcsszereplőnek megfelel-e az elkészült dokumentum. Az ellentmondás-mentesség ellenőrzés során azt vizsgáljuk, hogy nincsenek-e a dokumentumban egymásnak ellentmondó megszorítások, illetve adott szinten ugyanaz a funkcionalitás nem szerepe-e többször a dokumentumban. A teljességellenőrzés során állapítjuk meg, hogy a dokumentum tartalmazza-e az összes a felhasználók által igényelt követelményt. Megvalósíthatósági ellenőrzés során azt ellenőrizzük, hogy a követelményekben foglaltak technikailag kivitelezhetőek-e és, hogy a rendszerfejlesztés ütemtervei betarthatóak-e. A verifikálhatóság ellenőrzés során meggyőződünk arról, hogy a létezik-e olyan bizonyítható szabályrendszer, mely segítségével el lehet dönteni, hogy az adott követelményben leírtakat a szoftverrendszer teljesíti-e.

A validálás során használt eszközök közé tartozik a prototípuskészítés, mely egy végrehajtható modellnek tekinthető. Egy másik használható eszköz a követelmények felülvizsgálata, mely során egy a megrendelő és a fejlesztő alkalmazottjaiból álló vizsgáló csapat nézi át alaposan, több szempontból is figyelembe véve az összes követelményt. A verifikálhatóság bizonyítására egy módszer, ha elfogadási teszteseteket tudunk készíteni az összes követelményhez.

## A követelmények fontossága

A követelmények meghatározása, elemzése és karbantartása a megvalósítandó komplex szoftverrendszerek életciklusában egy kritikus fontosságú feladat, hisz ez a szakasz segít megérteni, és hogy a megrendelő mit is vár el a készítendő rendszertől. A feltárt követelmények alapján döntünk arról, hogy a rendszer megvalósítható-e és hogy megközelítőleg mennyibe fog kerülni a megrendelőnek. A szoftver komplexitása is becsülhető általa. Általában a szoftver specifikáció része a szerződésnek.

Hagyományosan a specifikáció köti össze a felhasználó igényeit és a fejlesztőket, így a kommunikációs szerepe is jelentős. Amennyiben a rendszer megrendelői mi magunk vagyunk, vagy a kommunikáció igen jó és gyakori a megrendelővel, illetve egyes metodikákat alkalmazva, a követelmények kifejtése a fejlesztés során több részletben, iteratívan történik.

## A követelmény feltárás, és kezelés problematikái

### A természetes nyelv problémái

A természetesen nyelven írt dokumentumok, mint amilyen lehet a felhasználói követelmények is, számos problémával rendelkeznek. Az egyik ilyen probléma az egyértelműség hiánya. Olykor nehéz a nyelvet pontosan használni, egy dolgot többféleképpen is leírhatunk, és a szavainknak is lehet számos jelentése. A szövegeink emellett terjengőssé is válhatnak, ezzel megnehezítve a lényeges információk kiszűrését. A folyó szövegben összemosódhatnak az egyes követelmények, illetve a különböző típusú, funkcionális és nemfunkcionális követelmények keveredhetnek így nehezebb ezeket elválasztani egymástól.

### A természetes nyelv problémáit kiküszöbölendő megfontolások

A követelmények megfogalmazásakor érdemes lehet minden követelményhez egy egyszerű magyarázatot fűzni, ami kifejti, hogy miért került be a megjegyzés, mi a szerepe annak. Ez nem csak érthetőbbé teszi az adott követelményt, de változásakor is segítséget nyújthat.

Érdemes egy szabványos követelményformátum elkésztése. Ami megadja, milyen formai és tartalmi szerkezettel adjuk meg az egyes követelményeket. Egyes ajánlásokban például a követelményeket kártyákra vették fel, minden egyes ilyen kártya egy darab követelményt tartalmaz, illetve tartalmazza az adott követelmény magyarázatát, más követelményekkel való kapcsolatát, azoktól való függését, illetve a követelmény forrását, azaz azt a személyt, akitől a követelmény ered. Így könnyen megtalálható az, akivel a követelmény változásakor érdemes lehet konzultálni. A formátum előírhatja továbbá a követelmény félkövér típussal való kiemelését, esetleg azt, hogy a felhasználói szintű követelmény tartalmazzon egy linket az azt részletező rendszerkövetelményre. Az ilyen kötöttebb, szabványos formátum megelőzheti az esetleges véletlen elhagyásokat, és könnyebben ellenőrizhetővé teszi magukat a követelményeket is.

Mint már egy korábbi szakaszban említettük érdemes lehet szétválasztani a szükséges illetve a javasolt követelményeket, például a „kell” vagy a „javallott” szavak felhasználásával és ezek következetes használatával.

A követelmény kulcsfontosságú részeit érdemes lehet kiemelni, félkövér, dőlt, vagy egyéb szövegformázási eszközökkel, ezzel elősegítve a követelmény későbbi olvasását, és az fontosabb információk hangsúlyozását.

Elkerülendőek a számítástechnikai szakzsargonok, de a rendszer felhasználási területének felhasználása elkerülhetetlen. Érdemes lehet az egyértelműség kedvéért fogalomszótárat használni.

Teljesség, ellentmondás mentesség, konzisztencia.

Analízis paralízis.

Változás.

Nyelvezet.

Elkülönítés, csoportosítás.

## 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.

## 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.

## Az üzlet és a követelmények modellezése UML segítségével

Az üzleti modellt a megrendelő és a rendszer kifejlesztésével megbízottak üzleti elemzői, közösen alkotják meg. Az üzlet egyes részeinek ábrázolására számos nem UML alapú módszer terjedt el, mint például a szervezeti diagram, mely a szervezeteken belüli alá- és fölérendeltséget, a vezetőségi hierarchiát modellezi, vagy a szervezeti folyamatmodellek, melyek üzleti feladatok végrehajtásához szükséges tevékenységek folyamatát fejezi ki. Vannak olyan üzleti modellek melyeken a piacépítést, bevételtermelést vagy az üzlet növelését lehet megtervezni. A különböző üzleti modellek az üzlet különböző aspektusait hivatottak mutatni.

Az informatikai rendszerek üzleti modellezésekor egy bevett eszköz az UML Használati-eset diagramja, amit ezen a modellezési szinten szoktak üzleti feladatdiagramnak is nevezni. Itt derítjük fel az üzleti szereplőket és az üzleti feladatokat, és ezek kapcsolatait. Az üzleti szereplőkről és a feladatokról érdemes szöveges leírást készítenünk. A feladat leírása tartalmazhatja a feladat definícióját, a fő célját és, hogy miért szükséges a rendszer, illetve az adott szereplők számára. Ezt a feladat küldetését áttekintő leírást célszerű közérthetően megfogalmazni. Mindemellett célszerű a feladathoz megadni egy szöveges forgatókönyvet, vagy más néven szöveges szcenáriót, mely a feladat végrehajtásának lépéseit, pontokba szedve, félig formális szövegként ábrázolja. Az üzleti feladatmodell egy másik gyakori diagramtípusa a tevékenységdiagram, mely a szcenáriók megadásának egy a szöveges forgatókönyveknél formalizáltabb módja.

A követelmény-feltárás során ezekből a diagramokból indulunk ki, illetve ezeket részletezzük. A finomítás során jelennek meg új diagram elemek, illetve az általánosítás, a kiterjesztés és a beszúrás kapcsolatok a használati eset diagram elemei között. Míg az üzleti feladatdiagramon az üzleti célok jelennek meg használati esetként, addig itt már jelöljük a rendszer által biztosított összes funkciót. A cél és a funkciók különbségét úgy lehetne szemléltetnem, hogy a felhasználó a rendszert nem keresi fel azért, hogy bejelentkezzen, ami lehet egy rendszerfunkció, de lehet az a célja, hogy a termékek között böngésszen. Az üzleti elemzésmodell, mely egy sztereotípiákkal ellátott osztály diagram, már alkalmazható az üzlet és az üzleti folyamatok során előkerülő összes szereplő, eszköz, és lépés a rendszer szempontjából szükséges virtuális lenyomatának modellezésére. A rendszerben szereplő dinamizmusok modellezésére a kommunikációs-, a szekvencia-, illetve az állapot diagram nyújtanak eszközt. A követelmény tervezés során a használati eseteket, illetve rokon funkcionalitásokat már gyakran elkezdjük csomagokba rendszerezni, ezek kapcsolatainak jelölésére kiváló eszköz az UML csomag diagramja.

## 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.

## Az agilis követelménykezelés és modellezés

A hagyományos IEEE 830 szabványban lefektetett, és ahhoz hasonló megfontolások alapján megírt követelménydokumentumok azt tárolják, hogy a rendszernek mit kell végrehajtania, és nem azt, hogy a felhasználónak mi a célja, aminek az elérésében a rendszer segíti őt. A terjedelmes több száz oldalnyi formális specifikációk, megnehezítik a teljes kép áttekintését és gyakran túl sok részletet próbálnak meg feltárni. A használati esetek már inkább a felhasználó céljait figyelembe véve készülnek, de a használati esetek is egy-egy nagyobb rendszerfolyamatot szemléltetnek, és a hozzájuk tartozó, a felhasználó és a rendszer kommunikációs sorát szemléltető fő és mellék forgatókönyvek a továbbiakban is a kelleténél túl sok részletet fed fel, illetve a feladatok priorizálására sem ad külön módot. Mivel a használati esetek egy folyamatot írnak le, ezért a megváltozásuk gyakran nagy kihatással járhat a rendszerre nézve. Mindemellett, a vázlatos használati eset diagram napjainkban is gyakran használt eszköz akár az agilis fejlesztések során is, mert egy jó vizualizációs eszköze lehet a felhasználói követelményeknek, viszont egyre inkább terjednek el és veszik át a helyét az egyszerű szöveges User Story-k.

### User Stories: A használati esetek agilis megközelítése

A User Story a használati esetekhez hasonlóan a felhasználó szemszögéből közelíti meg a rendszert, de nem egy folyamatot, hanem egy sokkal kisebb részt egy műveletét ír le.

A felhasználói sztori a felhasználó nyelvezetében, mondat formájában írja le az adott felhasználó rendszerrel kapcsolatos célját. Néhány szempontból hasonló a használati esetekhez, de leírásuk mindig informális és a felhasználó által könnyen értelmezhető. Az ilyen felhasználói sztorik annyira rövidek és tömörek, hogy többnyire egy kártyán vagy egy felragasztható jegyzetlapon is elférnek. A sztorik kisméretű a felhasználó számára értéket képviselő funkcionalitást írnak le. Mindig tartózkodnak a részletek említésétől, így nem vezetik se a felhasználó se a fejlesztő képzeletét. Fontos továbbá, hogy lehessen róluk beszélgetni, segítsék a párbeszédet a fejlesztő és a felhasználó között. Lényeges szempont, hogy a felhasználó is képes legyen meghatározni ezeknek a relatív fontosságát. Az ilyen szöveges leírások elkészítése és karbantartása külön előképzettség nélkül is könnyen elvégezhető. Míg a használati esetek létrehozását és karbantartását, bonyolultságuk miatt gyakran a fejlesztőknek kellett végezniük, addig a sztorikat már az üzleti oldal is nagyobb magabiztossággal elkészítheti.

A felhasználói sztorik kis méretének köszönhetően könnyebb velük áttekinteni a teljes rendszer összes értékes célját. Ezek a sztorik szoktak megjelenni a különböző backlogokban is, melyek a projectben jelenlévő összes hátralevő feladatot fontosságuk sorrendjében tárolják. A kis méret további előnye, hogy a megváltoztatásuk, és karbantartásuk is egyszerűbb. Amikor egy csapat oda kerül, hogy az adott sztorit elkészítik, tehát amikor szükséges, természetesen a sztorit is egy analízis során részletezhetik.

A kártyányi méret, és a kommunikáció segítsége mellett fontos szempont, hogy az adott érték megléte megerősíthető, illetve bizonyítható legyen, ezért a gyakorlatban minden sztorihoz tartoznak elfogadási kritériumok is, melyeket többnyire a Gherkin nyelv Given When Then formátumában adnak meg, melyről a későbbiekben részletesebben is írok.

A User Stories segítségével való modellezés során először felderítjük a rendszer fő céljait, ezeket a nagy horderejű átfogó sztorikat epikusoknak (epics) hívjuk, és a későbbiek során ezeket bontjuk fel részletesebb alsztorikra. Fontos, hogy a sztorik függetlenek legyenek egymástól, értéket képviseljenek. A méretük kezelhető legyen, tehát soha ne legyenek akkorák, hogy megnehezítsék a komplexitásuk és a fontosságuk megbecslését vagy a későbbi tervezésüket.

### A TDD, DDD, BDD hármas értelmezése

A Test Driven Developement (TDD) vagy a magyar terminológiát alkalmazva teszt vezérelt fejlesztés Kent Beck által megalkotott extrém programozási metodika (XP) egyik fontos eszköze. A TDD egy olyan fejlesztési ciklust ír le, ahol először egy a követelmények alapján megírt és még hibát mutató teszt megírását követően csak annyi programkódot írunk, amivel kielégíthetjük a teszt követelményeit, ezután egy kódújratervezéssel (Refactoring) javítjuk a kódunk és a programfelépítés minőségét. Ezt a három lépést ciklikusan egymás után hajtjuk végre mindaddig, míg a fejlesztés be nem fejeződik. Ez az eljárás elősegíti a jó programfelépítést és a magas tesztlefedettség révén a szoftver későbbi átalakítása is biztonságosabb lesz. A módszer egyes alkalmazói a követelményeket is így, beszédes tesztmetódus nevekben tárolták. A teszteknek meg van az az előnye, hogy a kód módosulását követik, és folyton friss és működő mintakódokat is tartalmaznak, ellenben a szakirányból érkező személyeknek az ilyen leírások nehezen olvashatóak.

A Domain Driven Design (DDD) azaz szakirány vezérelt tervezés egy Eric Evans nevéhez fűződő metodika, és szemléletmód, mely azt tanítja, hogy a technikai csapatnak is meg kel ismernie, és következetesen használnia kell a szoftver felhasználójának szakirányában használt terminológiáját. Fontosnak tartja az üzleti modellezést, és ezt olyan formában, hogy az az üzleti oldal számára teljes mértékben érthető és a szakirány szakértői által felügyelhető legyen. Fontos, hogy a szoftver fejlesztésében résztvevők tisztában legyenek a szakirány definícióival, és megfelelően kommunikáljanak a fejlesztésben érdekelt felek. Erre lehet egy jó módszer egy Domain Specific Language (DSL) alkalmazása. A DSL egy olyan redukált szókincsű és nyelvtanú nyelv, mely egy adott szakirány specifikus kifejezéseiből áll. Nem lehet egy ilyen korlátolt és egységesített nyelvvel bármit leírni, de alkalmas az adott szakirányhoz tartozó probléma precíz, következetes és a félreérthetőséget minimalizáló leírására.

A Behaviour Driven Development (BDD) azaz viselkedés vezérelt fejlesztés Dan North szoftverfejlesztő és Chris Matts üzleti elemző közös munkájából született. A modellezés középpontjába a rendszer viselkedése áll, tehát nem csak a szoftver feladatait írjuk le, hanem azt is, hogy milyen körülmények között milyen események hatására a rendszer hogyan reagál. Szemléletében az előző két említett módszertantól is sokat örökölt. A DDD-hez hasonlóan fontosnak tartja a szakirány nyelvezetének. használatát. A szoftver minőségének fenntartására BDD is teszteket használ, de ezek a tesztek nem egy-egy egység tesztje, hanem a rendszer integrációs és elfogadási tesztjei. Ezek a tesztek egy természetes nyelvű, de informális DSL segítségével szövegesen is le vannak írva, így a szakterület szakértői által ellenőrizhetőek, sőt maga az üzleti oldal is elkészítheti ezeket. A BDD a követelmények leírására a User Story eszközt használja, megírásuk közben pedig ajánlja az első szám harmadik személyben való kifejtését annak, hogy az adott szerepkörbe tartozó felhasználó milyen célból és milyen motivációktól vezérelve igényel a rendszertől egy adott funkcionalitást. A metodika alkalmazói körében a követelményekhez tartozó szcenáriók és elfogadási tesztek leírására és a kommunikációt segítő közös nyelv nyelvtanára a Gherkin DSL a legelterjedtebb.

### Gherkin nyelven leírt és „futtatható” követelmények.

Egy-egy szcenárió leírására alkalmas, kötött kulcsszavakból felépülő nyelv. A nyelvet a Cucumber nevű szoftverrel terjedt el először széles körben ahol a specifikáció írható le vele tömör formában, ebből készítik el általában az elfogadási teszteket, és a szoftver ellenőrzi is a készítendő rendszer megfelelőségét is ezek futtatásával, és az eredményük jelzésével. A nyelv előnye, hogy könnyen tanulható, a kisszámú kulcsszót bárki könnyen elsajátíthatja, és mégis hatékony kommunikációt segíthet elő az üzleti oldal, a fejlesztők, és a tesztelők között, de akár a felhasználó is átláthatónak találhatja. A kulcsszavak használata mellett a szövegeket strukturálni is kell, a Python programozási illetve a YAML adatstrukturáló nyelvhez hasonlóan, a sor elei behúzás jelzi a közös blokkba tartozó szövegeket, és azok hierarchiában elfoglalt helyét, és minden sor egy új kijelentést/gondolatot takar. Ha szükséges megjegyzéseket is írhatunk a kettős kereszt jel után. A nyelv további érdekessége, hogy a nyelv jelenleg a GitHub nevű verzió- és projectkezelő rendszeren keresztül nyílt forráskódú projectként fejlődik, és a kulcsszavakat számos nyelvre, köztük magyarra is lefordította a közösség.

A „Feature:” kulcsszót követően új blokkban szövegesen leírjuk az adott funkciót, képességet vagy üzleti igényt. Erre gyakran a User Story stílust használják.

A „Scenario:” (Forgatókönyv:) kulcsszó jelöli, hogy az utána következő blokk egy forgatókönyvet ír le. A forgatókönyvek leírására a nyelv védjegyévé vált Given-When-Then struktúra használatos.

A „Given” (Adott/Amennyiben) kulcsszó után, az adott sorban az adott forgatókönyv kontextusa, a rendszer kezdő állapota jelenik meg.

A „When” (Amikor/Ha) kulcsszó után egy feltételt, tevékenységsort vagy eseményt adhatunk meg.

A „Then” (Majd/Ha/Amikor) kulcsszó az elvárt végső állapot vagy történés jelölésére használatos.

Amennyiben a kezdő állapot vagy a tevékenységsor összetett, további sorokba az adott blokkhoz tartozást reprezentálóan egy szinttel beljebb húzva, „And” (És), illetve „But” (De) kulcsszóval kötve további részleteket adhatunk meg.

Teszt vázlatokat (Template) is megadhatunk. Ez egy sablon, amibe különböző helyekre, az oda tartozó értékek helyettesítődnek be. Ezt a Given-Then-When résznél használt kisebb-nagyobb jelek közé tett paraméternév, és egy Examples: résznél felsorolt paraméternév-érték táblázat megadásával érhetjük el. Ez a különböző teszteseteknek megfelelő bemenő értékeket reprezentálja.

A Background (Háttér) részbe azokat a Given részbe tartozó alapfeltételeket emeljük ki, amelyek az összes szcenárióra igazak, ezzel megelőzve a szóismétlést.

A Given-Then-When részekben hatékonyan önthetjük formába egy funkció követelményeit, és az elfogadási tesztjeinek tervezetét. Mindemellett közös nyelvként szolgál a fejlesztésben résztvevők között. Gyakran alkalmazzák azt a technikát, hogy a felek felteszik egymásnak a kérdést, mi van, ha az adott szcenárióban lecseréljük az adott kezdőállapotot, vagy a tevékenységet megváltoztatjuk, ezek vajon új szcenárióknak számítanak, esetleg egy hibás ágat tükröznek, hogy viszonyul ezekhez a változtatásokhoz a végállapot.

Bemutatok egy gyakran emlegetett példát.

User Story:

Mint egy vásárló visszahoztam a megvásárolt terméket

Forgatókönyv: Egyszerű áruvisszavétel

Adott egy vásárló, aki visszahozta a mikrohullámú sütőjét

És szeretné visszakapni a vásárlási összeget.

Amikor betér az egyik üzletünkbe

És felmutatja a vásárlást igazoló blokkot

Akkor visszaadjuk a pénzét.

Ennél az egyszerű forgatókönyvnél is számos résznél el lehet gondolkozni, mi van, ha a vevőnek nincs meg a blokkja, lehet másképp igazolni, hogy a terméket nálunk vette, mi van, ha a garanciaidőn túl hozta vissza az árut, mi van, ha az áru törött, ezekre az esetekre milyen eljárás vonatkozik. Mi van akkor, ha az alapállapotot módosítjuk, és egy nehéz hűtővel, vagy egy már beszerelt eszközzel például egy falra felszerelt televízióval, van a vevőnek problémája, ki gondoskodik az áru az üzletbe való visszajuttatásáról, a mi cégünk vállalja a szállítást, esetleg a cégünk szerelői kivizsgálják a terméket a helyszínen és, ha lehetséges javítják is a problémát, ezek mind olyan kérdések, amiket az üzleti oldallal kell megvitatni. Ellenben vigyázni kell, hogy a gyakori paraméter és and használat, az egyértelmű előnyein túl, egyesek számára áttekinthetetlenné és magát a leírást terjengőssé teheti. Fontos, hogy minél magasabb szinten, mindig csak a fontos részleteket kiemelve fogalmazzunk.

A nyelvhez kapcsolódó követelmény- és elfogadási tesztkezelő rendszerek számos platformon elérhetőek. Az alap Ruby nyelvet támogató Cucumbernek is elérhető számos platformra változata, illetve léteznek olyan szoftverek is, amik némiképp sajátságos szemlélettel dolgoznak, mint az RSpec, sőt az adott problémára egyes szoftverek alternatív nyelveket használnak a Gherkin helyett. A .NET környezetben az egyik legelterjedtebb ilyen a Cucumberhez hasonló szoftver a Visual Studioba integrálódó SpecFlow, de említésre méltó továbbá a NBehave, az BNSpec, a Concordion, a BDDfy is.

A Követelmény formátum sablonok fejezet megfelelő részénél a nyelv további szemléltetése is megtalálható.

## Követelmény formátum sablonok

### Felhasználói követelmény sablon

SORSZÁM (pl:2.3.1) Követelmény neve, nagyobb betűméret és félkövér formázás.

A követelmény rövid leírása, a lényeges részek félkövérrel kiemelve

Magyarázat: tömör magyarázata az adott funkciónak, dőlt betűs szedésben

Szerző: a követelmény készítőjének neve, elérhetősége

### Rendszerkövetelmény sablon

A szoftver neve félkövérrel szedve

|  |  |
| --- | --- |
| Funkció | A funkció neve / néhány szavas leírása |
| Leírás | A funkció néhány mondatos leírása |
| Bemenet | A rendszerfunkcióhoz szükséges bemenő adatok |
| Forrás | A be és a származási helyük leírása. |
| Kimenet | A szolgáltatás eredménye |
| Cél | A szolgáltatás célja |
| Művelet | A végrehajtandó műveletek leírása |
| Előfeltétel | A funkció meghívásának előfeltétele |
| Utófeltétel | A funkció eredményeként előállt utófeltételek |
| Mellékhatás | A funkció környezetére gyakorolt hatása |

### Formális használati eset szcenárió sablon

A folyamatleírások a használati eset lépéseinek listába szedett szöveges leírása.

|  |  |
| --- | --- |
| **Name** | Az adott használati eset neve. |
| Summary | Egy rövid összefoglaló a használati eset céljáról. |
| Rationale | A használati eset és a használatának a folyamat egyes körülményeinek kifejtése. |
| Users | A használati esetben résztvevő szereplők felsorolása |
| Preconditions | A használati eset előfeltételei |
| Basic Course of Events | A folyamat helyes ágának leírása, számozott lépéspontokba szedve. |
| Alternative Paths | Az egyes kibővítési pontokhoz tartozó folyamatirányok kifejtése. |
| Postconditions | A folyamat végső kilépési feltételei. |

### User Story sablon

A felhasználói sztorik leírása alapvetően lehet kötetlen, de napjainkban elterjedt sablonja az:

Én mint <felhasználói szerepkör> képes vagyok <egy tevékenységre> ezzel elérhetem az <üzleti értéket képviselő célomat>

Ugyanez angolul:

As a <role> I can <activity> so that <business value>

vagy

As a <role>, I want <a feature> so that <benefit>.

Ez a sablon azért olyan közkedvelt, mert külön részt szentel a „Miért” kérdésnek, így sokkal tisztább és érthetőbb a fejlesztésben résztvevő különböző személyeknek az adott sztori fontossága. Komplexebb több felhasználói szerepkört érintő leírásánál ajánlott áttérni az egyes szám harmadik személyre.

### Gherkin sablon

Feature:

Az igényelt funkció néhány soros leírása

Background:

Given a funkcióhoz tartozó összes szcenárióban közös kezdőállapot

Scenario:

Given az adott szcenariohoz tartozó alapállapot <megadható\_parameter\_1>

[{And|But} opcionális további állapotok]\* <megadható\_parameter\_2>

When rendszer esemény, <megadható\_parameter\_3>

Then végső történés, állapot vagy kiváltott folyamat <megadható\_parameter\_4>

Examples:

| megadható\_parameter\_1 | megadható\_parameter\_2 | …

| érték\_1 a paraméter\_1-hez | érték\_1 a paraméter\_2-höz | …

…

## 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.

## A szövegbányászatról általánosságában

Az emberek, már a kezdeti, ősi civilizációkban is, a szóbeli mellett, jellemzően írásbeli szövegek segítségével tárolták, és adták át egymásnak az ismereteiket. Napjainkban a rögzített tudásanyagainknak jelentős hányada egyszerű szöveges dokumentumokban található. Ezt a feltevést támasztják alá, többek között a Merill Lynch elemzései is, melyek becslése szerint az üzleti információk körülbelül 85%-a található strukturálatlan, illetve gyengén strukturált szövegekben. Az általunk kezelt szövegek növekvő arányban digitálisan tárolt dokumentumok.

Így talán nem meglepő, hogy a szövegek, és főképp a bennük lévő információk kezelésének hatékonyabbá tétele napjaink egyre fontosabbá váló informatikai tevékenysége. Az egyik tudományág mely ezzel foglalkozik, a szövegbányászat.

A szövegbányászatot definiálhatjuk úgy, mint szöveges adatokon végzett feldolgozási és elemzési tevékenység, melynek célja a dokumentumban rejtett információk feltárása, azonosítása, és elemzése. A szövegbányászat interdiszciplináris szakterület, mely olyan informatikai eszközök mellett, mint a gépi tanulás és a hatékony algoritmusok, a matematika és a nyelvészet eszközeit is felhasználja.

A szövegbányászat két nagy alaptípusa a keresés és a rendszerezés. A keresésnél kiválasztjuk azokat a dokumentumokat, ahol egy adott keresőkifejezés előfordul, míg a rendszerezésnél valamilyen kategóriákba vagy előre nem definiált csoportokba soroljuk azokat. A szövegbányászat főbb feladattípusai is ilyen jellegű feladatokat, vagy ezek kombinációját hajtják végre céljaik elérése során.

Ezek a főbb feladattípusok a kereséstámogatás és információ-visszakeresés, az információkinyerés, az osztályozás, a csoportosítás, az összegzéskészítés, a kivonatolás, a válaszkereső rendszerek, a szövegelemzés, és a napjainkban egyre inkább tért nyerő webes tartalomkeresés. Ezek közül csak a dolgozat szempontjából leginkább hangsúlyos információkinyerést fogom mélyebben részletezni.

## Szövegbányászati előfeldolgozás

Számos szövegbányászati feladat megoldható már létező adatbányászati eszközökkel és algoritmusokkal. Ehhez a szöveges adatokat úgy kell transzformálni, olyan alakra kell hozni, hogy ezek a bejáratott eljárások lehetőleg hatékonyan képesek legyenek működni rajtuk. Ezt a transzformációs lépést szokták előfeldolgozásnak is nevezni. A folyamat végeredménye a dokumentumot reprezentáló modell. Az előfeldolgozás egységesítési, formalizációs és normalizációs feladatokat is tartalmaz.

## Szövegbányászati modellalkotás

### Az ismertebb dokumentum reprezentációs modellek bemutatása

A legelterjedtebb modellek jellemzően valamilyen numerikus objektumok. Ez számos előnnyel jár. Az egyik előny a kisebb tárolási méret. Ugyanis ha a szavakat karakterenként letároljuk, és jellemzően egy-két bájtos egy karakter, akkor nagyobb helyet vennénk igénybe, mint a numerikus tárolásnál, ahol szavanként egy darab 2 vagy 4 bájtos számmal számolunk. Mindemellett, a számok használatának van egy másik jelentős előnye, méghozzá az, hogy matematikai műveleteket-, és transzformációkat hajthatunk végre az ilyen modelleken. Ráadásul a modellben a matematikai eszközökkel való munka elősegíti a dokumentumok hatékony kezelését.

Azt, hogy ténylegesen milyen modellt és adatábrázolást használunk, befolyásolja a megoldandó feladat típusa. Keresés jellegű feladatoknál egy megfelelő szóelőfordulás táblázat is nagy szolgálatot tehet, míg a rendezés jellegű feladatoknál összetettebb dokumentum összehasonlító módszerekre van szükség.

A modellalkotásnál használt három nagy matematikai elméleti megközelítés a halmazelméleti, az algebrai, illetve a valószínűségelmélet alapú. A halmazelmélet alapú modellek jó szolgálatot tehetnek az egyes keresőrendszerekben, hisz kereséskor minden dokumentumra fennáll, hogy része az eredményhalmaznak vagy sem. Az algebrai modellben a dokumentumokat olyan algebrai objektumokként reprezentáljuk, mint a vektor vagy a mátrix. Ezeket algebrai műveletekkel össze is hasonlíthatjuk, ezért ezek már használhatók rendszerezési feladatok megoldásánál is. A legelterjedtebb megvalósítása a vektortér modell és annak változatai. A valószínűségi modellben maguk a dokumentumok valószínűségi események által reprezentáltak, míg a kapcsolataik feltételes valószínűségi becslések eredményei. Ezek mellett léteznek komplex függvénytani eszközöket alkalmazó spektrális szövegbányászati modellek is.

### A vektortér modell

A vektortér modellben hatékonyan meg lehet határozni a dokumentumok távolságát, illetve hasonlóságát. A szövegbányászatnál gyakran élünk különböző intuitív heurisztikákkal, melyek meghatározzák az ezeket felhasználó eszközök felhasználhatósági körét és korlátait. A vektortér modellnél azt jelentjük ki intuitív módon, hogy azokat a dokumentumokat tekintjük hasonlónak, melyek szókészlete átfedi egymást, és ennek a hasonlóságnak a mértéke arányos az átfedés mértékével. A modell egy sokdimenziós vektortérben, vektorokkal reprezentálja a dokumentumokat. A vektortér egyes dimenzióit a dokumentumgyűjtemény egyedi szavai adják. Tehát egy-egy dokumentum a szavaiból álló vektor, abban a vektortérben, ahol az egyes szavak a teret kifeszítő vektorok. A dokumentumgyűjteményt egy szó-dokumentum mátrixszal reprezentáljuk. Az egyedi szavak összessége a szótár vagy más néven lexikon.

Általában a teljes szótár csak egy kis része fordul elő egy-egy dokumentumban, ezért a szó-dokumentum mátrix ritka. Mindemellett az egyedi szavak száma meglehetősen nagy, akár több százezres vagy akár milliós nagyságrendet is elérheti. Ezért a mátrix méretét nyelvtechnológiai, illetve matematikai eszközökkel érdemes lehet csökkenteni. A vektortérmodell másik nehézsége az alaphipotézisből származik, miszerint a szavak megléte, illetve száma adja a dokumentumok hasonlóságát. De nem tér ki és nem is tárolja a szavak sorrendjére se a szövegen belüli pozícióira vonatkozó információkat. Ezért hívják szózsákmodellnek is. Ez a modell a feladatok nagy részében jól használható, egyszerű és hatékony megoldást nyújt. A szósorrend elvesztéséből való hátrányát sokszor más modellekkel való együttes alkalmazásával küszöbölik ki.

### A szó-dokumentum mátrix jellemző súlyozási sémái

A legegyszerűbb módszer, ami csak a szó dokumentumbeli esetleges meglétét jelöli, a bináris reprezentáció, ahol is a mátrix egy adott dokumentumot reprezentáló oszlopvektorában egy adott szóhoz tartozó sorban nullát írunk akkor, ha az adott szó nem szerepel a dokumentumban és egyet, ha igen, és nem számít az, hogy hányszor; ez az információ elvész. Egy másik egyszerű módszer, ha a súly maga az előfordulások száma. Itt viszont felvetődhet a kérdés, miszerint tényleg lineáris-e a függés a szó adott dokumentumra vonatkozó relevanciája, és az adott szó előfordulásainak száma között. Tehát egy jóval gyakrabban előforduló szó tényleg ugyanilyen mértékben fontos-e. Az esetek nagy többségében a válasz egyértelműen: nem. Sokkal nagyobb megkülönböztető értéke van, ha az adott szó előfordul a dokumentumban, mint annak, hogy az tízszer vagy százszor fordul elő. Ezt kiküszöbölendő kezdték el használni a logaritmikus súlyozást. Ezen egyszerű súlyozási sémák nagy hátránya, hogy nem veszik figyelembe az egyes dokumentumok hosszát. Pedig feltételezhetően egy rövidebb dokumentumban egy szó bizonyos számú előfordulása fontosabb, mint az ugyanekkora számú előfordulása egy jóval hosszabb szövegben. Így a bonyolultabb súlyozási függvények figyelembe veszik a szó dokumentumbeli előfordulási gyakoriságát is. Ezt a gyakoriságot az angol irodalomban term frequency kifejezéssel illetik, és a szakirodalmakban gyakran TF-nek rövidítik. Ezen elven alapuló jellemző megvalósítások a gyakoriság alapú, illetve a normalizált logaritmikus súlyozások. Ez a megközelítés már közelebb áll a valósághoz, de nem veszi figyelembe azt, hogy vannak olyan szavak, melyek minden dokumentumban egységesen sűrűn fordulnak elő és nincsenek semmilyen összefüggésben az adott dokumentumok tartalmával. Az angol terminológiát átvéve ezeket stopszavaknak hívjuk. Ezért a szó fontosságát befolyásolhatja a teljes gyűjteményben való előfordulásainak száma is, amit szoktunk gyűjteménytámogatottságnak is nevezni, vagy angolul collection frequency az elterjedt megnevezése. Fontos lehet továbbá az előfordulások eloszlása a korpuszon, ugyanis a koncentráltan kevés dokumentumban, de azokon belül nagy számban szereplő szavak valószínűleg fontosabbak is az adott dokumentumokra nézve, illetve nagyobb a dokumentumok közti megkülönböztető szerepük is. Ha nk-val jelöljük a tk szót tartalmazó dokumentumok számát, és N pedig a dokumentumok száma a korpuszban, akkor a dokumentum gyakoriságot a df = nk/N képlettel számolhatjuk. Ez a szám megadja az adott szó ritkaságát, azt, hogy a szó megléte és annak száma mennyire különbözteti meg a dokumentumokat egymástól az adott korpuszban. Az idf (inverse document frequency) súlyozási sémában ennek inverzével számolunk. Maga a dokumentumgyakoriság inverzét számoló egyik elterjedt súlyozási függvény:

idf(tk) = log(N/nk)

A leggyakrabban használt tf-idf (term frequency and inverse document frequency) súlyozást úgy kapjuk, hogy a vektortérmodell szavakat reprezentáló tengelyeit az adott szavak idf által megadott relevanciájával arányosan súlyozzuk.

Tehát a szó-dokumentum mátrix adott dik súlya kiszámítható a

dik= fki \* idf(tk) képlettel

E mellett számos más képlete és módosulata létezik, amikre most nem térek ki.

A tf-idf súlyozás értéke magas lesz a nagy megkülönböztető képességű, adott dokumentumra gyakori, de a korpuszra ritka szavaknál. Alacsonyabb lesz a korpuszban gyakoribb vagy az adott dokumentumban ritkább szavaknál, és elhanyagolhatóan alacsony, akár zérus az olyan szavaknál melyek az egész korpuszban gyakran fordulnak elő.

### Módok a szöveg felbontására, illetve a szótár felépítésére

Mielőtt felírhatnánk a választott modellbe a dokumentumainkat, számos az azt megelőző előfeldolgozási lépésre lehet szükségünk. Az első ilyen lépés a reprezentációs egységekre való bontás. Ez leggyakrabban egyszerű szavakra való bontás, de egyes esetekben, mint például a kivonatolásnál a dokumentumok bekezdésekre történő, illetve mondat szintű felbontására is szükség van. Ezután a vektortér modell használata előtt meg kell határozni a szótár szavait is. Az összes egyedi szó szövegekből való kinyerése után célszerű lehet ezek egy kis fontosságú, de esetenként sok szót tartalmazó részét elhagyni, ezzel jelentősen csökkentve az indexelt szavak számát és a vektortér méretét.

### A strukturális szegmentálás

Dokumentumaink szövege számos hierarchiaszintbe sorolható. Ilyen strukturális egységek lehetnek például a kötet, rész, fejezet, szakasz, pont, illetve a bekezdés, a mondatok és a szavak. Ezek és az ezekből összeállítható dokumentumtérkép automatikus meghatározása egyes fájlformátumokból kinyerhető információk alapján viszonylag egyszerű feladat is lehet, míg például az egyszerű szövegekből nehézkes, esetenként koránt sem egyértelmű feladat lehet.

### Mondatokra bontás

Szövegeink mondatokra való bontása automatizálható, de ez nem triviális feladat. Kézenfekvő lenne egyszerűen a mondatvégi írásjeleket nézni, és ezek szerint elszeparálni a mondatokat. Ilyenkor viszont álmondathatárokkal is találkozhatunk, hisz például a pont is előfordulhat egy mondat közepén, rövidítéseknél, sorszámoknál, dátumokban, vagy akár IP-címekben is. A feladat megoldására általában szabály alapú döntési algoritmust alkalmaznak, mely az ilyen téves mondathatárokra utaló, vagy ezeket cáfoló jelekre és tulajdonságokra tartalmaz szabályokat. Ezeket a szabályokat próbáljuk illeszteni a szövegeinkre. A tulajdonságok lehetnek felszíniek, mint például a jelet követő szó nagy kezdőbetűs e vagy sem, esetleg a jel egy szó közepén van-e stb. Az algoritmus az illesztés mellett kereshet gyakori álmondathatárokat tartalmazó listákban is, mint például a rövidítéslisták. Az ilyen listák nyelvenként eltérőek lehetnek, és időnként a lista karbantartására, frissítésére is szükség lehet. A szabályok tartalmazhatják, hogy a vélt határtól milyen pozícióra elhelyezkedő és milyen jellegű szavakra, vagy pontosabban tokenekre vonatkozik, és szokás megadni hozzájuk egy súly értéket is, hogy amennyiben egy vélt mondatvégre több szabvány is illeszkedik, a nagyobb súllyal rendelkező juthasson érvényre. Legyen szabályrendszerünk bármilyen komplex vagy kifinomult, előfordul, hogy az algoritmus nem képes jól dönteni, mert a mondat, vagy az egymás után elhelyezkező mondatok értelmezésével lehet csak jól meghatározni a mondatok valódi határát.

### Tokenizálás

A token egy bizonyos karaktersorozat konkrét előfordulása, míg típusnak nevezzük az azonos tokenek osztályát. Így a típusok összessége alapján állítható elő a szótár. Ez a tokenizációs lépés minden lényeges szövegbányászati feladatnál, mind a keresés, mind pedig a rendszerezés jellegűeknél, egyik fontos előfeldolgozási lépés. Hisz míg például a keresésénél ezekre illesztünk, és információkinyerésnél ezek potenciális információt hordozó entitások lehetnek, addig a dokumentumok rendszerezésénél, ezek olyan attribútumok lehetnek, melyek segítségével mérhetjük az egyes dokumentumok összetartozását, hasonlóságát. Ezek a feldolgozásunk legkisebb részelemei. A gyakorlatban sokszor egyszerű szavak, de mint a fenti definícióból látható lehetnek más szövegegységek, és karaktersorozatok, ritkán és szélsőséges esetekben egész dokumentumok is.

A tokenek alatt leggyakrabban mégis elszeparált szavakat értünk. Az ilyen tokenek előállítása elsőre egyszerű is lehet, hisz elég csak az írásjeleket elhagyva, a szóközöknél elválasztani a szöveget. De számos további kérdés is felmerül, mint, hogy hogyan kezelhetjük a kötőjellel elválasztott vagy aposztrófot tartalmazó szavakat, egyben vagy külön tokenekként. Kérdés továbbá, hogy az egyéb írásjeleket tartalmazó szavakat, mint a „kukac” és a „pont” karaktert tartalmazó e-mail címekre milyen szabályt használjunk. Továbbá hogyan kezeljük azokat a jeleket, amilyen az informatikusok többségének egyértelműen külön fogalmat reprezentáló C++, itt sem hagyható el a két pluszjel, hisz úgy már egy merőben más fogalmat kapnánk. Kérdés továbbá, hogy miként kezeljük a tulajdon neveket, különböző együttesek vagy csoportok több tagból álló neveit vagy a konkrét eseményt jelölő dátumokat. Természetesen az előbbi kérdésekre több helyes válasz is elképzelhető a feladatunktól függően, illetve a nyelvi és a felhasználási környezet is nagyban befolyásolhatja, hogy konkrétan milyen módon tokenizálunk.

Egy alternatív megoldás is létezik, amit karakter n-gramm módszernek nevezünk. Ekkor a szöveget n hosszúságú karaktersorozatokra bontjuk. Például: a „karakter 4-gramm” szöveget 4-grammokra bontva a következő eredményt kapjuk: kara, arak, rakt, akte, kter, ter˽, er˽4, r˽4-, ˽4-g, 4-gr, -gra, gram, ramm. Ez az eljárás jól használható nyelvek meghatározásakor, vagy más egyszerű osztályozási problémánál, esetleg olyan jellemzően távol-keleti nyelveknél szótárépítésre, ahol nincsenek a különböző szavak szóközökkel elválasztva egymástól. De nem célszerű a használata az európai eredetű nyelveknél, olyan feladatoknál, ahol a szótár elemeinek értelmes egységeknek kell lenniük.

A tokenek típusai lesznek a nyelvi elemzés alapvető szövegreprezentáló egységei, és különböző szűrő eljárások után a típusok építik fel magát a szótárat is.

### Stoppszószűrés

Egyes szavak, illetve tokenek nagy számban fordulnak elő a dokumentumgyűjtemény dokumentumaiban és nincsenek szoros kapcsolatba azok témájával; ilyenek a névelők, a határozószavak és a névutók. Ezek a stopszavak. A stopszószűrésre vagy azok feldolgozás közbeni kiküszöbölésére láttunk egy matematikai jellegű módszert, a td-idf súlyozást. Tehát a stopszavakat a tokentípusok előállítása után, különböző dokumentumgyakorisági adataik alapján határozzák meg, majd ezeket a gyakorlatban esetenként manuálisan ellenőrzik, és ezek egy részét, a feladattól függően elhagyhatják. Napjainkban például keresésénél, esetleg maximum néhány tízes vagy akár tíz alatti számú szót hagynak el. Míg az elhagyások száma egyes osztályozási és csoportosítási feladatokban ez több százas vagy akár ezres nagyságú is lehet.

Ennek az az oka, hogy a keresésénél az elhagyott stopszavak szerepelhetnek ismert kifejezésekben, különböző irodalmi vagy más művészeti művek címeiben, és elhagyásukkal ezekre való keresés is nehézkessé válna. Az indexállományok tömörítése, illetve a megnövekedett, egyre olcsóbb tárkapacitás együttesen elérték, hogy az idők folyamán egyre kevesebb és kevesebb stopszót kelljen elhagynunk. Mindeközben a rendezési, és más pontos egyezést nem igénylő feladatoknál, ezek a szavak, konkrét jelentés nélkül, csak megnehezítenék a feldolgozást, nélkülük nagymértékben lecsökkenhet a reprezentációs modell mérete. Gyakran szabják az adott korpuszra az elhagyható szavak listáját, így biztosítva a lehető leghatékonyabb és mégis biztonságos feldolgozást.

### Lemmatizálás és szótövezés

A stopszószűrés mellett léteznek nyelvészeti eszközök is a szótárak szűrésére és méretének redukálására. Az alapötlet az, hogy a legtöbb nyelvben vannak a szavaknak különböző módosult alakjai. Ha ezeket a különböző alakokat egy közös alakként tárolnánk, egyes feladatokban elhanyagolható szóalak információ elvesztése révén, nagymértékben redukálhatnánk a modell méretét. A csökkentés mértékét befolyásolhatja az adott nyelv morfológiájának gazdagsága. Például amíg ez a mérték az angol nyelvnél megközelítőleg 40-70 százalék között lehet, addig a magyarnál elérheti akár a 90%-ot is. A lemmatizálás, és a szótövezés is ilyen közös, úgymond kanonikus alakra hozó módszerek. A szó módosult alakjait a toldalékok, mint a képző, a jel és a rag adják. Ezek a toldalékok a szó előtt és a szó után is állhatnak, körül is vehetik a szótövet, vagy a magyartól idegen módon, be is ékelődhetnek a szótőbe. A közös alakra hozásnál ezeket a toldalékokat igyekszünk valamilyen módon leválasztani az adott szóról.

A lemmatizálás közben a szó normalizált szótári alakját, azaz a lemmáját határozzuk meg. Nyelvészeti motivációjú eljárás közben mindig értelmes szótári alakot hozunk létre. A lemmatizálás nem egyértelmű, ezért az eredmény több szó is lehet. Például a falunk szó szótári alakja a falu, de lehet a fal szó is.

A szótövezés egyfajta szócsonkolást eredményez. Nem az a célja, hogy értelmes szótári szó jöjjön létre, sokkal inkább az, hogy a kialakított új szó a valódi szövegben is megtalálható legyen. A szótövező eredménye függ az adott algoritmustól.

Hasonlítsuk össze néhány példán a két megközelítést. A munkát és a munkám szavak lemmája a munka, szótöve algoritmustól függően a munka, vagy a munká. A ló lemmája a lovak, lovát és maga a ló szavaknak, de ezek szótöve a választott szótövező függvényében lehet a lo vagy a ló is.

## Információkinyerés

Az információkinyerésnél (Information Extraction - IE) a fő célunk nagy mennyiségű szövegből kigyűjteni a legfontosabb információkat. Tesszük ezt olyan formában, hogy azt később akár egy relációs adatbázisba is beírhatjuk. Tehát a strukturálatlan adatokat kívánjuk valamilyen struktúrában összefoglalni. Az információkinyerés napjaink meghatározó szövegbányászati kutatási iránya, hisz kiválóan alkalmas lehet nagy mennyiségű emberi munka kiváltására. Az adatok strukturált formába öntésével segíti a folyamatosan növekvő mennyiségű információinkat könnyebben kezelhető és jobban áttekinthető, jobban ellenőrizhető és feldolgozhatóvá tenni. Egyik dinamikusan fejlődő altípusa a nyelvközi információkinyerés (Cross-Language IE), melynél az adatokat több különböző nyelvű szövegekből is összeszedjük és táblázatba öntjük, majd elég csak a táblázatfejléceket lefordítani a kívánt nyelvre.

Az információ-kinyerés nagymértékben feladatfüggő megoldásokat kíván, mert többnyire csak előre rögzített típusú elemeket vagyunk képesek a szövegekből hatékonyan kinyerni. Fontos lehet ismernünk a felhasználási szakirány egyes jellemzőit ahhoz, hogy kideríthessük, hogy az adott feladat szempontjából mik a leginkább fontos attribútumok, amiket a szövegből ki szeretnénk gyűjteni, és azokat milyen módon és formában lehet célszerű a felhasználók számára prezentálni. A megoldásunk továbbá függeni fog az alkalmazási terület jellemző korpuszától is.

### Az információkinyerés összehasonlítása más főbb feladattípusokkal

Az információkinyerés a rendezési alaptípusokkal szemben, nem dokumentumokat akar valamilyen szempont szerint besorolni valamilyen kategóriába, mint az osztályozás, továbbá nem célja dokumentumok valamilyen szempont szerinti csoportosítása sem.

A keresési alaptípusba tartozó információ-visszakereséssel szemben nem lokalizálja a felhasználó számára az információt, és nem választja ki a dokumentumkollekció egy kifejezés alapján releváns dokumentumait. Ezzel szemben kiválasztja a feladat szempontjából fontos dokumentumokat, ezeket önmagukban elemzi, és a kivont és formába öntött információkat juttatja vissza a felhasználónak. Az információkinyerő rendszerek feladatfüggőek, míg az információ-visszakeresés problémája és megoldása lehet általános. Így a keresőrendszerek kialakításához kevesebb felhasználási szakterülettel kapcsolatos háttérismeretre van szükségünk. A keresés gyorsabb, és az eredményét a felhasználó manuálisan tekinti át. Így, ha esetlegesen hibát vét a rendszer annak nincs különösebb kockázata. Ellenben kevesebb munkát igényel a felhasználótól az információ kinyerő által feldolgozott tömörebb adatmennyiség áttekintése. Az információkinyerő rendszerekben gyakran kell a többi feladattípus megoldásait felhasználni.

## Az információkinyerés fontosabb részfeladatai

A probléma megoldása során számos, olykor egymásra épülő feladatot kell végrehajtanunk. Ezek rendre a névelem felismerés, a kereszthivatkozás azonosítás, a szereplők azonosítása, a szereplők közti relációk azonosítása és az események illesztése.

### A névelem felismerés

A névelem, vagy angolul Named Entity, egy, a valóságban előforduló, névvel ellátott objektum-egyed megnevezése. Az ilyen egyed lehet egy személy tulajdonneve, egy telefonszám, egy e-mail cím vagy bármilyen más azonosító. Ezek felismerése olykor egyszerű reguláris kifejezésekkel történik, de főleg a tulajdonnév-felismerésnek nevezett alfajában bonyolultabb módszerekre is szükség lehet. A tulajdonnév-felismerés problémakörébe nem csak a tényleges tulajdonnevek felismerését szoktuk érteni, hanem különböző együtt kezelendő tokensorozatokat is ide értünk. Például egyes főneveket, mint igazgatóság, vagy bevásárlóközpont esetenként érdemes lehet a név részeként tekinteni. Manapság a névelem-felismerésre gyakran különböző gépi tanulókat alkalmaznak. Ezek a mesterséges intelligencia eszközeit alkalmazó módszerek, a névelemeket számukra új, de azért a tanulóadatokhoz hasonló tulajdonságú szövegeken is eredményesen fedezhetik fel.

További példák lehetnek névelemekre például hírekben a személyek, helyek, konkrét időpontok, szervezetek, egyes eszközök konkrét megnevezése, de biológiai szövegekben lehetnek akár gének, fajok, fehérjék megnevezései is. A specifikációs szövegekben is előfordulhatnak névelemek, különböző cégek vagy konkrét személynevek formájában, de ennél jellemzőbb, hogy különböző felhasználói szerepköröket kell jelölnünk.

### A szereplők azonosítása

A szövegbányászatban szereplőkön többnyire, bizonyos előre definiált sablonoknak megfelelő névelemek előfordulásait értjük. Az ilyen felderítendő szereplők lehetnek például vásárlók, termékek; orvosi szövegekben például a betegségek tünetei vagy a gyógyszerek. További gyakori felderítendő szereplők lehetnek például a helység, és a szervezet. Gyakran a szereplők konkrét szövegbeli előfordulásait meg is szoktuk jelölni, a szerepnek megfelelő címkével. Például LOC jelölést kapnak a szövegben a konkrét helységnevek, ORG címkével látjuk el a szervezetek neveit, és a személynevek a PER címkét kapják.

A specifikációs szövegekben szereplőkön különböző felhasználói kategóriákat értünk. Ezek a szereplők inkább mint kategória osztályok vagy szerepkörök jelennek meg a szövegekben, semmint egyedek közös sablonra illeszkedő sokasága, ezért megtalálásukra is némiképp eltérő módot célszerű keresni.

### A kereszthivatkozások azonosítása

Ez egy olyan lépés, melyben az azonos egyedre utaló tokensorozatokat és névelemeket rendeljük össze. Ennek egyik válfaja az anaforafeloldás, ami a különböző névmásoknak és utalószavaknak a megfelelő névelemhez való rendelését jelenti.

### A szereplők közti relációk azonosítása

Ebben a lépésben az a célunk, hogy az előre meghatározott, és felderített szereplők közti kapcsolatokat ismerjünk fel.

Például üzleti hírekben vásárló és vásárolt áru, esetleg az áruhoz tartozó ár meghatározása vagy orvosi folyóiratokban lehet egy adott betegség, és az azt gyógyító gyógyszer összerendelése. Specifikációkban ilyen feladat lehet egyes aktorok és az általuk végezhető használati esetek összerendelése, esetlegesen egyes használati esetek és szereplők egymás közti viszonyainak feltárása.

### Események felfedése és illesztése részfeladat

Eseményen többnyire egy bizonyos történést vagy cselekvést értünk. Az ilyen események szövegeinkben gyakran igék formájában jelennek meg, és fontos lehet az azonosításuk. Miután az összes eddigi részfeladatot elvégeztük, rendelkezünk előre definiált események konkrét dokumentumbeli előfordulásával, illetve az azokhoz köthető szereplőkkel, és köztük fennálló különböző kapcsolatokkal. Ebben a lépésben pedig megoldást adunk arra, hogy ezek alapján különböző kérdésekre válaszolhasson a rendszer.

Ilyen kérdések lehetnek például, hogy egy konkrét piaci szereplő milyen termékeket vett egy adott évben, vagy a szoftverkövetelmények területén az, hogy egy adott használati esetben mely szereplők vehetnek részt, vagy éppen fordítva, egy adott aktor milyen feladatokat végezhet az adott szoftverrendszeren.

## Az információkinyerés során előforduló főbb nyelvészeti problémák

Az információkinyerő rendszer a működése érdekében szükség szerint számos nyelvtani jellegű problémát old meg.

A keresett eseményeket vagy funkciókat gyakran cselekvést kifejező szavakkal, általában igékkel írják le. Így fontos lehet ezeknek a szövegben való azonosítása. Előfordulhat, hogy egy szóalakhoz tartozik főnévi értelmezés is, mint például a nyúl szó is lehet egy cselekvést leíró ige, de főnévi értelmében már egy állatot jelöl. Egyértelmű, hogy a két különböző jelentésnek megfelelően, az ilyen szavak különböző feldolgozást is igényelnek. Egyes esetekben az ige helyett névszói frázis fejezi ki az adott cselekvéseket. Ezen feladatokban lehetnek segítségünkre a morfológiai, illetve szófaji elemzések.

Az igéknél gyakran el kell dönteni, hogy azok jelen-, jövő-, vagy múlt- idejűek. Fontos lehet továbbá, hogy az adott ige módja kijelentő vagy feltételes. Emellett a felderített információkról el kell dönteni, hogy állító vagy tagadott értelemben szerepelnek-e.

Általában igaz az, hogy a különböző szereplők, események és relációk a nekik megfelelő helyen, a rájuk jellemző mondatrészben találhatóak. Így fontos lehet a mondatok szerkezetének meghatározása. Az alany, az állítmány és a mondat további bővítményei, mint például a tárgy és a határozók közti kapcsolatok ismerete is hasznos lehet.

Gyakran fordul elő, hogy az eseménnyel kapcsolatos névelemeket különböző beékelődött főnévi frázisok szeparálják el az eseményt jelző igétől. Fontos lehet ezek felismerése.

## A szabály és a statisztika alapú információkinyerési megközelítések összevetése

A legegyszerűbb mód az entitások, a relációk és a különböző események kinyerésére, ha különböző nyelvi mintákat definiálunk, és ezek segítségével végezzük a kinyerést. Ilyen minta lehet például, ha egy tetszőleges hosszú nagybetűs szót a Kft. rövidítés követ, abból arra következtetünk, hogy a nagybetűs szó egy szervezet típusú névelem. Ennek a megközelítésnek a fő előnye, hogy meglehetősen nagy pontossággal és hatékonyan képes, az előre definiált típusok felfedezésére. Hátránya azonban az, hogy csak a szabályrendszer által lefedett típusokat deríti fel, és hogy a szabályrendszer folyamatos karbantartást igényel. A gyakorlatban az ilyen szabályokkal általában nehéz komplexebb problémákra jól működő megoldást adni. Ennek a fő okai, hogy a nyelvekben ugyanazt az információt többféleképpen is kifejezhetjük, minden lehetőség lefedésére irreálisan nagy és nehezen kezelhető szabályrendszert kéne alkotnunk, különben megkockáztatjuk, hogy a jelenlévő hasznos információnak csak a töredék részét fedezzük fel. A szövegek nyelvtani elemzésének felhasználása nagymértékben képes javítani a módszer használhatóságát. A szabályrendszer ilyen módú általánosítására kereteket alkalmazunk. A keret továbbá leírhat egy szereplőt, vagy akár szereplők egymáshoz való viszonyát. Egy keret meghatározhat nyelvi mintákat, megadva például olyan nyelvtani tulajdonságokat, mint a szófaj, az eset, vagy, hogy az adott szó egyes vagy többes számban szerepelhet. A keretek illesztésénél nem kell, hogy a vizsgált mondatban, a keret által meghatározott összes szerep jelen legyen; ez nagymértékben megnöveli a keretek rugalmasságát. Léteznek általánosan felhasználható metakeret adatbázisok, mint például a Berkley egyetem által létrehozott FrameNet.

A szövegbányászat során a legtöbb problémára lehet találni statisztikai eszközöket alkalmazó megoldást. Ez alól az információkinyerés sem kivétel. Gyakran alkalmazunk különböző gépi tanulókat, melyeket előre megjelölt, felcímkézett szövegeken tanítunk be. Ezek képesek az ismert tanító címkék alapján, a számukra új teszt-, és a végleges működési adatokon, az eddigieknek megfelelő, vagy valamilyen azokhoz hasonló új elemek felderítésére. Napjainkban ez a megközelítés egyre szélesebb körben terjed, ugyanis a szabály alapú rendszerek számos problémáját kiküszöböli. Az ilyen statisztikai alapú módszer általános, problémafüggetlen megoldást biztosíthat. Persze a rendszer jó működésére a továbbiakban is szükség van a jó minőségű, és lehetőleg a felhasználási környezethez illeszkedő tanítóadatra. Emellett a gyakorlatban a felidézése, tehát, hogy milyen mértékben találja meg a jelenlévő információt, és a pontossága, azaz, hogy mekkora mértékben tipizálja helyesen az információkat az esetek nagy részében jobb, mint a hasonló, de szakértői szabályokat használó rendszereké.

### A statisztikai megközelítések által használt elterjedt reprezentációs modellek

Két elterjedt megközelítés létezik; az egyik tokenek szekvenciáit, jellemzően teljes mondatokat használ a modellek alapegységeként, ezt hívják szekvencia alapú modellnek, míg a másik megközelítés magukat a tokeneket tekinti alapegységnek, ezt hívják token alapú modellnek. Az első módszer célja, hogy teljes mondatok osztálycimke-sorozatát, azok mondatstruktúráját, képes legyen előre jelezni. A szakirodalomban ezt szokták strukturált predikciónak is nevezni.

A másik módszer, mely külön kezeli a tokeneket, képes lehet új, számára eddig ismeretlen tokenek címkéjének automatikus előállítására. Az adott token környezetében lévő szavakra vonatkozó információk itt sem vesznek el, legfontosabb jellemzőikkel, az adott szó jellemzővektorát bővíthetjük ki.

A szekvencia alapú modellek többnyire valamelyik Markov-modell, vagy hasonló statisztikai modellek alapján működnek, míg a token alapú megközelítés a problémát felügyelt gépi tanulók segítségével igyekszik megoldani.

## További szövegelemzési megfontolások

A szövegek nyelvtani elemzését és annak a szoftverek tervezésben való használatáról először Russell J. Abbott 1983-ban publikált Program design by informal English descriptions című művében tesz említést. Módszerének lényege, hogy a probléma szöveges megfogalmazását alapul véve annak az egyes főneveiből határozza meg a rendszer objektumosztályait, és azok attribútumait, a szöveg igéiből pedig a rendszertől elvárt szolgáltatások, illetve funkciók származhatnak. Abbott szerint az angol szöveg analízise során felfedezett tulajdonnevek konkrét objektumokra, a köznevek osztályokra, a cselekvést kifejező igék metódusokra, a létigés szerkezetek generalizációra, a „has an” jellegű szerkezetek aggregációra utalhatnak.

Az ilyen jellegű szöveg analízist az európai hadászati és légi iparban is használatos HOOD metodológia is alkalmazza. Booch is Abbott munkájából indult ki és fejlesztette azt tovább saját módszerének a megalkotása során. Az OMT módszer kezdeti verzióiban is, a fejlesztés első lépéseként, szövegelemzéssel határozták meg a rendszer főbb osztályait, majd a későbbiekben az OMT-II egy megelőző lépésként a használati esetek használatát ajánlotta a kiinduló szövegek létrehozásának megkönnyítésére.

A módszer egyik hibája, hogy automatikus használata nagy mennyiségű fals osztályt eredményez. Ezért a Rational Unified Process már a szövegben előforduló szakterületi fogalmak jelöli ki fő osztályoknak. Ez a módszer is számos felesleges osztályt jelöl ki, ellenben nem foglalkozik olyan kérdésekkel, mint, hogy az adott szereplőknek legyen osztály megfelelőjük a rendszerben, és melyeknek ne.

Robert C. Martin Tiszta kód című művében, amikor a helyes és beszédes elnevezéséről értekezik, utal arra, hogy az osztályok nevei mindig főnevek, vagy főnévi szerkezetek, de semmiképpen sem igék. Kerülendőnek tartja az olyan általános osztályneveket, mint az Adat, a Menedzser vagy az Info, ezeknek a kifejező ereje igen csekély, félreértésekre adhat alapot, hogy gyakran környezetfüggő a jelentésük, ezért legfeljebb egy másik főnévvel együtt alkothatnak jól használható osztálynevet. A specializált osztálynevek megalkotására pedig az ősosztály főneve elé írt melléknevek használatát ajánlja. A tagfüggvények neveinek pedig igék vagy igei kifejezések használatát javasolja. A melléknevek jelenléte, konstansokra vagy az egymás utáni melléknevek enumeráció jellegű adatszerkezetre utalhatnak.

Martin a névválasztásnál a megoldástartomány neveit részesíti előnyben, de kiemeli, hogy ha nem létezik ilyen, akkor a feladattartomány, azaz a szakirány kifejezései is hasznosak lehetnek, hiszen az ilyen szavak használatakor a szakirány szakértőivel is érdemes lehet konzultálni, ha a fejlesztés során problémába ütközünk.

Véleményem szerint, a szövegek nyelvi elemzése, kiegészítve különböző szakiránnyal kapcsolatos szótárak használatával és a kisebb megkülönböztető képességű szavak kiszűrésével, egy jó eszköz lehet, a rendszer főbb osztályainak azok kapcsolatainak feltérképezésében.

## Az irodalomkutatás alapján leszűrt tanulságok

### Egy követelményelemzést támogató eszköz főbb tulajdonságai.

A követelménytervezés megvalósíthatósági tanulmány elkészítésének részfolyamatát az eszköz külön támogatja, de lehetőséget biztosít a megvalósíthatósági jelentés dokumentumként való feltöltésére. Érdemes lehet, lehetősséget biztosítani a projecthez a project sablonjában a project felelőse által meghatározott módon bármilyen dokumentum feltöltésére.

A követelmények feltárására és modellezésére az szoftvereszköz számos technikát támogat, kezdve az olyan hagyományosabb eszközökkel, mint a használati esetek módszere, vagy mint a napjainkban divatos User Story eszköz. A követelmény feltárás megkönnyítésére online elérhető interjús űrlapokat is biztosít. Az etnográfiai megfigyelést külön nem támogatja, de annak a végső dokumentumát képes eltárolni. A követelmények elkészítését, különböző nézetekkel segítheti. Például érdemes lehet egy képernyőn megjeleníteni a felhasználói követelményt, a hozzá tartozó rendszerkövetelményeket, és az ezekhez kapcsolható modellrészeket.

A követelményspecifikációs fázist különböző akár személyre is szabható szoftverdokumentum sablonokkal, és azok egy részének generálásával képes támogatni.

A validáció támogatása egy érdekes megközelítés lehet, a követelmények különböző hasonlósági szempontok alapján, akár szövegbányászati eszközökkel történő csoportosítása, és a csoportokba tartozó követelmények együttes mutatása. Ezzel a valamilyen a szempontból összetartozó követelmények között esetleg könnyebben észrevehető, ha a követelmények között több is ugyanazt írja le, és ezek egymásnak esetleg ellent mondanak.

A verifikáció támogatására a Gherkin nyelvű szcenárió megfogalmazás szerepel, mint eszköz. A Gherkin nyelvű szcenáriók szövegesen megfogalmazott elfogadási teszteknek tekinthetőek.

Követelmények kezelését és követését az eszköz úgy igyekszik támogatni, hogy a különböző összetartozó követelmények és modellelemek közti kapcsolatot tart fenn, és jelzi, ha az adott elemmel összefüggő elemek valamelyikében változás következett be.

### Projektszótárak és szövegszínezés

A projectekben érdemes létrehozni, és karban tartani egy olyan ontológiát, azaz értelmező szótárat, ami a rendszerben és annak szöveges dokumentumaiban előforduló legfontosabb szavakat és azok magyarázatát tartalmazza. Érdemes lehet ezeket a szavakat a szövegben megjelölni, és amennyiben a felhasználó ezt igényli, a rendszer megjeleníthetné, az adott szóhoz tartozó értelmezést. A rendszer arra is módot adhatna, hogy a szövegben kijelölt szót az arra jogosult felhasználó felvegye, és az értelmezését megadja. Ilyen módszerrel továbbá meg lehetne jelölni a szövegben előforduló szereplőket, használati eseteket és entitásokat is.

### Az információ kinyerés lehetséges megfontolásai az adott feladatkörben

A fejlesztés során a felhasznált szövegbányászati keretrendszer segítségével elvégzem a szükséges előfeldolgozási lépéseket. Ezek között kiemelt fontosságú a szófaji elemzés (Part Of Speech Tagging), melyet az irodalomkutatás További szövegelemzési megfontolások alfejezetében tárgyaltak szerint kívánok felhasználni. A felhasználók által karbantartott szakirányú szótárak és a tf-idf súlyozás segítségével kialakítom a dokumentum valószínűleg legfontosabb szavainak halmazát, majd ennek a segítségével tesz a szoftver ajánlatot a szerepkörök, az entitások, a főbb osztályok és esetleg az azok közti kapcsolatok modellre történő felvételére. A szoftver ilyen jellegű predikciós képességét egy lehetséges továbbfejlesztés során tovább lehet finomítani, ha a szoftver felhasználói, a saját tapasztalataik alapján szabályokat, illetve kereteket vehetnek fel, a kiválasztás megkönnyítésére. Egy másik későbbi fejlesztési lehetősség, nagy mennyiségű és jó minőségű kész projectekből származó mintaadat megléte mellett, gépi tanulók bevetése.

### Mik lehetnek egy modern CASE eszköz főbb tulajdonságai.

Véleményem szerint a jövő CASE eszközeinek a csoportos munka támogatására kell törekedniük, illetve arra, hogy a szoftverfejlesztés minél nagyobb területét lefedjék. Úgy tegyék mindezt, hogy egymással a lehető legnagyobb összhangban működnek. Emellett a továbbiakban is törekedniük kell a munka, minél teljesebb, minél hatékonyabb, a lehető legkevesebb emberi beavatkozást igénylő, segítésére.

### A szöveg-kiegészítésnél alkalmazható megfontolások

A különböző szó és szöveg kiegészítések, vagyis kód kiegészítések az elterjedt programozási környezetekben már alap funkcionalitás. Ellenben a napjainkban elterjedt szövegszerkesztők ezt a funkcionalitást nem támogatják. Valószínűleg ennek oka abban keresendő, hogy míg a programozási nyelvekben a kulcsszavak száma korlátozott, és az értelmesen behelyettesíthető kódrészek számát is korlátozza az adott kontextus, addig a természetes nyelveknél a felhasználó által behelyettesíteni kívánt szavak kiválasztása már sokkal összetettebb feladat.

A természetes szövegek szavainak kiegészítésének egy kézenfekvő módja lenne, hogy egy adott nyelv összes szavát tároljuk és ezek közül a szavak közül, az adott szóból leütött karakterek alapján, abc sorrendben felajánljuk a szavakat. A módszer előnye, hogy mindenképpen képes megfelelő szavakat ajánlani, ellenben ennek a megközelítésnek számos hátránya van. A nagy mennyiségű szószedetből nehézkes és lassú a kereséses, ráadásul az ilyen szótárakat létre is kell hozni, karban is kell tartani, ami nagy mennyiségű emberi többletmunkát igényelhet. A megközelítés további problematikája, hogy az abc sorrend miatt előfordulhat, hogy a felhasználónak szinte az egész szót végig kell írnia, amire elérkezik a kívánt szóhoz, ezért nem lesz kellően hatékony a szókiegészítés.

Egy másik megoldás az, ha az adott szövegben már előforduló szavakból ajánljuk fel az eddig leütött karaktersorra leginkább illeszkedő szavakat. Ez a megközelítés előnyös, mert nem igényel szótárkarbantartást és egy jóval kisebb szómennyiségből ajánl fel szavakat, ami gyorsabb és hatékonyabb kiegészítést tesz lehetővé. A szakmai szövegekben az irodalmi jellegűekénél gyakoribb a szóismétlés, így itt még inkább előnyös lehet ez a megközelítés. A módszer hátránya, hogy a szövegben még elő nem fordult szavakat nem képes felajánlani. Ezt a hátrányt azzal próbálják ellensúlyozni, hogy a rendszer több dokumentumot is képes beolvasni, és így létrehoz egy saját szótárat, ezáltal egy frissen kezdett dokumentumban is képes ajánlatot tenni a kiegészítésre. Ezt a szótárat, a frissen írt szövegekből nyert új szavak felhasználásával, a kiegészítő folyamatosan frissítheti. A módszer egy másik hátránya, különösképpen a gazdag morfológiájú nyelveknél, mint amilyen a magyar nyelv is, hogy a rendszer a már bevitt szómódosulatokat ajánlja fel kiegészítésre, ami gyakran olyan szótöredéket is tartalmaz, amit a felhasználó vissza kell, hogy töröljön. Hasonló módon működik a LibreOffice Writer szövegszerkesztő aktuálisan legfrissebb 4.3.4-es verziójában megtalálható opcionálisan aktiválható szókiegészítő is, ami az eddig a szövegszerkesztővel megnyitott szövegek szóhalmazából ajánlja fel az eddigi szókezdemény elejére leginkább illeszthető szavak közül az abc-ben először előfordulót.

Véleményem szerint, a rendszeremben két módszer előnyeit fogom egyesíteni, illetve az abc szerinti sorrendet, egy a szavak fontosságát a dokumentum aktuális felépítése alapján megadni képes heurisztikára cserélem. Továbbá szavak különböző alakjainak felajánlása előtt, azok szótövét is felajánlom.

A dokumentum alapszótára az eddigi projectek és szövegek beolvasott egyedi szavaiból fog állni, az eddigi gyakoriságuk szerint súlyozva. Azon szavak súlyát, melyek az adott projectben is előfordulnak, megemelem. Ezután a projecthez tartozó szakirányú szótárban is szereplő kifejezések súlyát tovább emelem. Esetleg az írás környezetében lévő szavakat is tovább súlyozom, azon megfontolás alapján, hogy az egymáshoz közeli szövegeknek valószínűleg hasonló a fő témájuk és ez a használt szavakban is megjelenhet. Amikor az adott szöveg egy sablon szerint strukturált, akkor a rendszer csak az adott helyre értelmes javaslatokat tegye. Például egy User Story-ban ahol egy adott helyen csak aktor szerepelhet, ott a rendszer csak a szereplőket ajánlja fel. A súlyok meghatározása után a szavaknak meghatározom a szótöveit, illetve az egy tő alá tartozó szavak leghosszabb közös résszavát. A szótövek, megkapják az alá tartozó szavak súlyösszegét. A rendszer először a súlyuk alapján sorba rendezett és a felhasználó által elkezdett szókezdeményre leginkább illeszkedő szótövek közül ajánlja majd a legnagyobb súlyút, illetve módot ad a lista további szótöveire való ugrásra. A felhasználó újabb karakter leütésével a kiegészítési ajánlatot figyelmen kívül hagyva folytathatja az írást vagy egy a behelyettesítést kiváltó billentyű lenyomásával, a kívánt szótő beírásra kerül, és felajánlja a rendszer a leghosszabb közös résszót, itt is dönthet úgy a felhasználó, hogy egy másik szóváltozatot ír be, vagy elfogadja a leghosszabb közös résszó behelyettesítését. Ez után a rendszer az eddig használt szavak közül fogja ajánlani a legmagasabb súlyút. Reményeim szerint ezzel a módszerrel minimalizálhatom a követelmények leírásához a felhasználó által leütni szükséges karakterek számát.

### A modell elkészítését segítő módszerek.

A diagramelemek felvételének számos módja lehetséges, a legelterjedtebben használt módszerek egyike a már elkészült modellelemek közül húzd és ejtsd módszerrel, az adott diagramelem, diagramra való felvétele. A másik lehetőség, egy eszköztárról kiválasztható típusú diagramelem sablonjának kitöltése és a felvétele a diagramra. Ilyenkor viszont felmerülhet a kérdés, hogy az így felvett diagramelem új modellelemet is jelent, vagy egy már létező modellelem adott nézőpontja.

A modell- és diagramelemek felvételekor érdemes lehet használni az előző fejezetben tárgyalt strukturált szöveg kiegészítéséhez hasonló eszközt. Így elősegítvén a projectben szereplő elnevezések konzekvens használatát. A diagramelemek felvételekor, a kiegészítés hatására a rendszer fel is ajánlhatja, hogy az aktuálisan felvett diagramelemet összekapcsolja a neki megfelelő modellelemmel. Amennyiben ezt a javaslatot a felhasználó elutasítja, új modellelem kerül létrehozásra, vagy, ha ez nem valamilyen szabály megsértését eredményezné, a felhasználót figyelmezteti a rendszer. Amennyiben a felhasználó nem a szövegkiegészítést használja, ez az ellenőrzés természetesen a diagramsablon elmentésekor ugyancsak lefut.

A kiegészítés a különböző modellszinteken fellelhető többletinformációkat is felhasználhatja. Ilyen lehet például, ha van egy adott osztály példányának, egy szekvencia diagramon kívánunk üzenetet küldeni, a rendszer kiegészítésre felajánlhatja az adott osztály metódusait, és amennyiben a felhasználó nem ezek közül választ, hanem új változatot vesz fel, a rendszer felveszi azt a modellen, így az osztály diagramon is megjelenhet.

# Már létező hasonló jellegű programok vizsgálata

A következőkben két elterjedt modellező eszközt vizsgálok meg. A vizsgálat során elsősorban azt figyelem meg, hogy az adott szoftver miképpen támogatja a követelmény folyamat egyes lépéseit, a modellezést, illetve a csoportos munkavégzést.

## Sparx Enterprise Architect

A Sparx cég Enterprose Architect (EA) nevű eszköze kimagasló UML támogatással rendelkezik és számos programozási nyelven készült kódbázissal képes szinkronban tartani a modellt. A vizsgálat során a termék 11.1-es verzióját használtam.

### Követelményfolyamat támogatása

A rendszer lehetőséget nyújt a követelmények modellelemként való felvételére, és azok egymáshoz való kapcsolására, illetve azok készültségi fokának, illetve komplexitásának megadására. A követelmények felvételének egy másik eszköze a használati esetek, melyet a rendszer támogat. A használati esetekhez leírást és szcenáriót is megadhatunk. A szcenáriók megadásához használhatunk természetes nyelvet. A rendszerben az adott használati esetekhez felvehetünk továbbá aldiagramokat, többek között aktivációs-, szekvencia-, kommunikációs- és állapot átmenet diagramot. Az eszköz módot ad a felhasználói felület prototípusának megrajzolására és a követelményekhez való kapcsolására.

### A modellezés támogatása

A rendszer kiemelkedő tulajdonságokkal rendelkezik, a modellezés terén, ugyanis számos rendszermodellező eszköz mellett, mint a SysML, az UML 2.5 szabvány szerinti modellezést is elősegíti.

### Csoportos munka támogatása

Az EA a projecteket külön fájlkiterjesztésű (.eap) fájlokban tárolja. Lehetősség van arra, hogy ezeket a fájlokat, külső verziókezelő rendszer segítségével ossza meg a csapat egymással. Ez csak kis teamek esetében jelenthet megoldást, és nehézkes az ütközések feloldása, amennyiben többen egyszerre szerkesztették az adott projectfájlt. A szoftvert továbbá be lehet úgy konfigurálni, hogy a project adatait ne a saját fájljába, hanem egy külső adatbázis kezelő rendszerben tárolja, ezzel lehetősséget adva a csoportos diagramkészítésre.

## Visual Paradigm

Egy másik elterjedten használt modellező eszköz a Visual Paradigm (VP), az EA-nál kevesebb modelltípust és programozási környezetet támogat, de a modellezés támogatásához egy jóval ergonomikusabb eszköz. A modellelemek felvételére a hagyományos eszköztáras fogd és vidd módszer használható. Kész elem jobb felső sarkában lévő nyílra kattintva is létrehozhatunk új elemeket.

### Követelményfolyamat támogatása

Az EA-hoz hasonlóan itt is lehetősség van, a követelmények diagramelemként való felvételére. Itt egy alapsablon és néhány kiegészítő kapcsolat is adott, amik a követelményelemzéshez segíthetnek. A követelményeket a rendszer alap szinten ellenőrzi is a követelmény elemek helyességét, például szól, ha nincs az adott elem csomagba téve, vagy a szövege nem szerepel a project szótárában, esetleg, ha az adott követelmény nem kapcsolódik a modell egyetlen más eleméhez sem. Ez az eszköz is támogatja a használati esetek módszerét, de a szcenáriók leírására az EA által támogatottak mellett User Storykat is használhatunk, melyeket külön priorizálhatunk és akár sprintekhez is rendelhetünk. A rendszer lehetősséget ad továbbá az Abbotéhoz hasonló szöveges analízishez, mely során egy egyszerű szöveges fájlban jelölhetünk ki részeket és címkézhetjük fel azokat szereplőnek, használati esetnek, osztálynak, vagy egyszerű projectszótárbeli elemnek.

### A modellezés támogatása

A diagramok megrajzolására számos ötletes eszközt ajánl, a hagyományos eszköztárról húzd és vidd módszer mellett, az eszköztár elemére duplán kattintva tudunk ugyanabból az elemtípusból egymás után többet is felvenni. Amennyiben egy kész diagramelemre kattintunk, az elem körül számos elem jelenik meg, amik az elemből kihúzva az eddigi diagramelemhez kapcsolt, a kiválasztott elemnek megfelelő elemet hoznak létre. Ez némileg kényelmesebb megoldás lehet az EA jobb felső nyilacskájánál. A modellelemeket összekötő nyilak elhelyezése is némileg több módot ad, mint az Enterprise Architect.

### Csoportos munka támogatása

A csoportos munka támogatására a termékhez járó VPository nevű felhőszolgáltatást nyújtja a cég, melyre verziókezelő rendszerekhez hasonlóan, lehet feltölteni és leszinkronizálni a projectet. A konkurens változtatások kezelését is megoldja a rendszer. Mód van egy ehhez hasonló szerver beüzemelésére is, amennyiben a felhőszolgáltatást nem tartjuk megfelelőnek.

## Értékelés és az elkészítendő rendszer a két létező rendszer tükrében

Az Enterprise Architect egy kivalló modellező eszköz, de inkább a modellezés alsóbb szintjeinek a kóddal való szinkronizálásában emelkedik ki. (Például van saját Visual Studio és Eclipse beépülő modulja is, ami az egyszerű kódgenerálás és visszaolvasáson túl további funkciókat is nyújt). A Visual Paradigm ezeken a pontokon némileg alul marad, hisz csak a Java és a C++ nyelvet támogatja, de a diagramok elkészítése némileg komfortosabb vele, továbbá a csoportmunkára is ad egyszerűbb megoldást. Mindkét szoftver támogatja a modellből történő dokumentációgenerálást, de az ellenkező irányt már kevésbé támogatják. Ezek a szoftverek valamilyen módon a szoftverprojectek megvalósításának számos szintjét támogatják.

Az általam létrehozandó szoftver módot kíván adni arra, hogy a követelményeket szövegként is meg lehessen alkotni. Célja továbbá, hogy egy reszponzív webes alkalmazásként számos platformról, és eszközről kezelhető legyen. Mindemellett megoldást kíván nyújtani arra, hogy a felhasználók folyamatosan, közel valós időben láthassák egymás munkáját, így az eszköz jóval kényelmesebb és produktívabb kollaborációt tehetne lehetővé. Ellenben a rendszer nem kíván verseire kelni a két már létezőtesztelt eszköz összes funkcionalitásával, illetve a követelménykezelésnél mélyebb absztrakciókat sem kíván támogatni.

# A rendszerrel szemben támasztott követelmények

## Vázlatos felhasználói követelmények, illetve igények.

(VÁZLATOS)

A megvalósítandó rendszer egy követelmény specifikációs és követelmény analízist támogató intelligens, elosztott eszköz.

Az eszköz kell, hogy rendelkezzen webes felhasználói felülettel, mely lehetőleg minél több platformon helyesen jelenik meg.

Könnyen kezelhető és intelligens eszközöket biztosít a követelmény feltárás, elemzés és modellezés megkönnyítésére.

Módot kell adnia a különböző követelmény- és modell elemek egymáshoz való kapcsolására, és a meghatározott kapcsolatok segítségével történő navigálásra,

Esetleg jó lenne, ha lehetőséget biztosítana az összekapcsolt elemek egy azon nézetben való megjelenítésére.

Jó lenne, ha a rendszerben, különböző szótárak kezelésével lehetőséget biztosítana a rendszerben jelen lévő entitások azonosítására, esetleges szövegekben való kijelölésére, írás közbeni szöveg-kiegészítésére.

Jó lenne, ha a rendszerrel végzett munkát, minél előbb, akár a munkavégzés pillanatában, valós időben látnák a rendszer további felhasználói.

A rendszer módot biztosít a rendszer felhasználóinak különböző felhasználói szerepkörökhöz való rendelésére.

Jó lenne, ha a rendszer, az alap felhasználói szerepkörök finomhangolására is módot adna, az arra jogosultak számára.

A rendszer felhasználói interfészének törekednie kell egyes modern felülettervezési ajánlások betartására.

Jó lenne, ha a felhasználó által kijelölt elemek környékén megjelenő környezetfüggő menü átlátható módon tartalmazná a legfontosabb teendőket az adott elemen, ezzel elősegítve a kényelmes és hatékony kezelést. Az itt nem megjelenő menüelemek, egy a modern Office programokból megismert szalagmenüben kaphatnának helyet.

A rendszernek módot kell biztosítania a további bővítésének megkönnyítésére.

Jó lenne, ha egy követelmény megváltozása kijelölné felülvizsgálatra a kapcsolódó követelmény, és modell elemeket.

Jó lenne, ha le lehetne kérdezni, hogy mely követelmények lettek kifejtve az adott szinten, és melyek azok, amelyek még „érintetlenek”.

Az eltérő követelménytípusok különböző színű „kártyákon” jelenhetnének meg.

## A rendszerben megjelenő felhasználói szerepkörök

### A rendszer felhasználóinak használati eset nézete



### A rendszerben megjelenő, előre definiált felhasználói szerepkörök jellemzése

A felhasználó szerepköre megszabja, hogy az adott felhasználó milyen módon és milyen jogokkal férhet a rendszerhez. A különböző szerepkörök képviselői hasonló felhasználói felülettel rendelkeznek, de a számukra nem engedélyezett funkciók, nem jelennek meg előttük.

A **General Reader** (általános olvasó) van a hierarchia tetején és a legáltalánosabb jogokkal rendelkezik a rendszeren belül.

A **Software Architect**, a **Business Analyst**, illetve a **Convention Manager** az olvasó szerepkört specializálják, az ő funkcionalitását öröklik, illetve saját felelősségi körökkel bővítik. Tehát a General Reader lehetősségein felül további jogokkal, és lehetősségekkel rendelkeznek.

A **Convention Manager** a rendszerben megjelenő sablonokat, konvenciókat, és megkötéseket olvashatja.

A **Business Analyst** a felhasználói követelmények feltárására, és szerkesztésére specializálódott szerepkör.

A **Software Architect** a rendszerkövetelményeket, és rendszermodelleket állítja elő a magas szintű követelmények alapján.

A Convention Manager konkretizálása a **Company Contact**, aki a cégre vonatkozó általános konvenciókat szerkeszti, illetve a **Project Manager**, aki project szinten finomítja ezt. A **Project Manager** a Business Analyst leszármazottja is, így a magas szintű követelmények szerkesztésére is módja nyílik.

## Funkcionális követelmények

### A rendszer főbb felhasználói funkcióinak áttekintő használati eset nézete



A rendszer fő funkciói, az UML használati esetet modelljének OMG általi ajánlásainak megfelelően, kerettel el van határolva, a rendszer külső felhasználóitól.

A General Reader általános olvasási jogokkal rendelkezik, illetve megjegyzéseket is hozzáfűzhet a rendszer elemeihez.

A Software Architect kezeli a rendszerkövetelményeket, míg a Business Analyst a felhasználói követelményeket szerkeszti.

Míg a Convention Manager olvashatja a céges, illetve a project szintű megszorításokat, addig a Company Contact, illetve a Project Manager szerkeztheti is a felelősségi körének megfelelő konvenciókat.

A szerepkörök egy másik ága az Administrator, aki a felhasználókat, és azok szerepköreit és jogosultságait kezeli.

## Nem funkcionális követelmények

# A rendszer kialakítása során felhasználható technológiák és jellemzésük

## Szerver oldali technológiák

### ASP.NET

### Java

EE vagy Spring MVC esetleg valami egyszerűbb REST FW?

## A kliens oldali megjelenítés kezelése

### HTML5

SVG vagy Canvas

### CSS

## A kliens oldali dinamika kezelése

### JavaScript, és a TypeScript

## Kommunikációs technológiák

REST – a WinApi, illetve egy Javas megoldás;

SignalR

## Szövegbányászati csomagok

# Követelményanalízis

## A rendszer statikus modelljének elemzése

### Szerkezet áttekintő nézet

### Szerkezeti nézet

## A rendszer dinamikájának és folyamatainak elemzése

## Képernyő vázlatok

# Rendszerterv

## Képernyő tervek

## Csomagáttekintő nézet

## Részletes komponens leírások

## Részletes dinamikus nézetek

## Telepítési nézet és rendszerkörnyezet

# Az implementáció részletei

# Tesztelés

## Statikus tesztelés

## Dinamikus tesztelés

### Egység teszt

### Komponens teszt

### Integrációs teszt

### Rendszer teszt

### Átadási teszt

# Továbbfejlesztési lehetőségek

# A szakdolgozat tartalmi összefoglalója

# Irodalomjegyzék

[1]. **Sommerville, I.** *Szoftverrendszerek fejlesztése.* hely nélk. : Panem Kiadó Kft., 2007.

Ian Somerwille: [1] asdf Szoftver rendszerek fejlesztése

Vég Csaba: Alkalmazásfejlesztés a Unified Modelling Language szabványos jelöléseivel

Sike Sándor, Varga László: Szoftvertechnológia és UML

Tarczai Tünde: UML diagramok a gyakorlatban

Robert A. Marksimchuk, Eric J. Naiburg: UML földi halandóknak.

Tick D.: Szövegbányászat

Robert C. Martin: Tiszta kód

Eric E. Domain Driven Design Reference

http://www.informit.com/articles/article.aspx?p=29043&seqNum=9

http://hu.wikipedia.org/wiki/Toldal%C3%A9k

http://www.tankonyvtar.hu/hu/tartalom/tamop425/0046\_szoftverfejlesztes/ch07s04.html

<http://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requirements>

<http://www.slideshare.net/dimka5/agile-stories-estimating-and-planning>

<http://www.stellman-greene.com/2009/05/03/requirements-101-user-stories-vs-use-cases/>

<http://www.batimes.com/articles/user-stories-and-use-cases-dont-use-both.html>

<http://www.agilemodeling.com/artifacts/useCaseDiagram.htm>

<http://ryantablada.com/post/we-are-complicating-things-a-bit-too-much-%28tdd-and-ddd%29>

# Mellékletek