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](#_Toc418971380)

[1 Tartalomjegyzék 3](#_Toc418971381)

[2 Ábrajegyzék 7](#_Toc418971382)

[3 Célkitűzések 8](#_Toc418971383)

[4 Irodalomkutatás 9](#_Toc418971384)

[4.1 Bevezetés 9](#_Toc418971385)

[4.2 A Követelménytervezés alapfogalmai 9](#_Toc418971386)

[4.2.1 Szoftverkövetelmény 9](#_Toc418971387)

[4.2.2 Követelmények csoportosításai 10](#_Toc418971388)

[4.2.3 Felhasználói követelmények 10](#_Toc418971389)

[4.2.4 Rendszerkövetelmények 10](#_Toc418971390)

[4.2.5 Funkcionális követelmények 10](#_Toc418971391)

[4.2.6 Nemfunkcionális követelmények 11](#_Toc418971392)

[4.2.7 Szakterületi követelmény 11](#_Toc418971393)

[4.2.8 Szükséges és javasolt követelmények 12](#_Toc418971394)

[4.2.9 Kulcsfigura 12](#_Toc418971395)

[4.2.10 Verifikáció és Validáció 12](#_Toc418971396)

[4.2.11 Fogalomszótár 12](#_Toc418971397)

[4.3 A követelménytervezés folyamata 12](#_Toc418971398)

[4.3.1 Megvalósíthatósági tanulmány elkészítése 13](#_Toc418971399)

[4.3.2 Követelmények feltárása és elemzése 13](#_Toc418971400)

[4.3.3 Követelmény specifikációs folyamat 14](#_Toc418971401)

[4.3.4 Szoftverkövetelmények dokumentuma 14](#_Toc418971402)

[4.3.5 Követelmények kezelése 14](#_Toc418971403)

[4.3.6 Követelmény validáció 15](#_Toc418971404)

[4.4 A követelmények fontossága 15](#_Toc418971405)

[4.5 A természetes nyelv problémái 16](#_Toc418971406)

[4.5.1 A természetes nyelv problémáit kiküszöbölendő megfontolások 16](#_Toc418971407)

[4.6 Az üzlet és a követelmények modellezése UML segítségével 16](#_Toc418971408)

[4.7 A követelmények modellezése közben használt diagram eszközök 17](#_Toc418971409)

[4.7.1 A használati-eset diagram 17](#_Toc418971410)

[4.7.2 A tevékenység diagram 18](#_Toc418971411)

[4.7.3 Sztereotípiákkal ellátott elemzési osztálydiagram 18](#_Toc418971412)

[4.7.4 Szekvencia diagram 19](#_Toc418971413)

[4.7.5 Együttműködési diagram 19](#_Toc418971414)

[4.7.6 Állapotdiagram 19](#_Toc418971415)

[4.7.7 Csomagdiagram 20](#_Toc418971416)

[4.8 Az agilis követelménykezelés és modellezés 20](#_Toc418971417)

[4.8.1 User Stories: A használati esetek agilis megközelítése 20](#_Toc418971418)

[4.9 A szövegbányászatról általánosságában 21](#_Toc418971419)

[4.10 Szövegbányászati előfeldolgozás 22](#_Toc418971420)

[4.11 Szövegbányászati modellalkotás 22](#_Toc418971421)

[4.11.1 Az ismertebb dokumentum reprezentációs modellek bemutatása 22](#_Toc418971422)

[4.11.2 A vektortér modell 23](#_Toc418971423)

[4.11.3 A szó-dokumentum mátrix jellemző súlyozási sémái 23](#_Toc418971424)

[4.12 Információkinyerés 24](#_Toc418971425)

[4.13 További szövegelemzési megfontolások 24](#_Toc418971426)

[4.14 Az irodalomkutatás alapján leszűrt megfontolások 25](#_Toc418971427)

[4.14.1 A követelményelemzést támogató eszköz főbb tulajdonságai. 25](#_Toc418971428)

[4.14.2 A specifikációs szöveg és a specifikációs modell közti híd megteremtése 25](#_Toc418971429)

[4.14.3 Mik lehetnek egy modern CASE eszköz főbb tulajdonságai. 26](#_Toc418971430)

[5 Már létező hasonló jellegű programok vizsgálata 26](#_Toc418971431)

[5.1 Sparx Enterprise Architect 26](#_Toc418971432)

[5.1.1 Követelményfolyamat támogatása 26](#_Toc418971433)

[5.1.2 A modellezés támogatása 27](#_Toc418971434)

[5.1.3 Csoportos munka támogatása 27](#_Toc418971435)

[5.2 Visual Paradigm 27](#_Toc418971436)

[5.2.1 Követelményfolyamat támogatása 27](#_Toc418971437)

[5.2.2 A modellezés támogatása 27](#_Toc418971438)

[5.2.3 Csoportos munka támogatása 28](#_Toc418971439)

[5.3 Értékelés és az elkészítendő rendszer a két létező rendszer tükrében 28](#_Toc418971440)

[6 A rendszerrel szemben támasztott követelmények 28](#_Toc418971441)

[6.1 Vázlatos felhasználói követelmények, illetve igények. 28](#_Toc418971442)

[6.2 A rendszerben megjelenő felhasználói szerepkörök 30](#_Toc418971443)

[6.2.1 A rendszer felhasználóinak használati eset nézete 30](#_Toc418971444)

[6.2.2 A rendszerben megjelenő, előre definiált felhasználói szerepkörök jellemzése 30](#_Toc418971445)

[6.3 Funkcionális követelmények 32](#_Toc418971446)

[6.3.1 A rendszer főbb felhasználói funkcióinak áttekintő használati eset nézete 32](#_Toc418971447)

[6.4 Nem funkcionális követelmények 33](#_Toc418971448)

[7 A rendszer kialakítása során felhasználható technológiák és jellemzésük 34](#_Toc418971449)

[7.1 Szerver oldali technológiák 34](#_Toc418971450)

[7.1.1 ASP.NET 34](#_Toc418971451)

[7.1.2 Java 34](#_Toc418971452)

[7.2 A kliens oldali megjelenítés kezelése 34](#_Toc418971453)

[7.2.1 HTML5 34](#_Toc418971454)

[7.2.2 CSS 34](#_Toc418971455)

[7.3 A kliens oldali dinamika kezelése 34](#_Toc418971456)

[7.3.1 JavaScript, és a TypeScript 34](#_Toc418971457)

[7.4 Kommunikációs technológiák 34](#_Toc418971458)

[7.5 Szövegbányászati csomagok 34](#_Toc418971459)

[8 Követelményanalízis 35](#_Toc418971460)

[8.1 A rendszer statikus modelljének elemzése 35](#_Toc418971461)

[8.1.1 Szerkezet áttekintő nézet 35](#_Toc418971462)

[8.1.2 Szerkezeti nézet 36](#_Toc418971463)

[8.2 A rendszer dinamikájának és folyamatainak elemzése 36](#_Toc418971464)

[8.3 Képernyő vázlatok 37](#_Toc418971465)

[9 Rendszerterv 37](#_Toc418971466)

[9.1 Képernyő tervek 37](#_Toc418971467)

[9.2 Csomagáttekintő nézet 37](#_Toc418971468)

[9.3 Részletes komponens leírások 38](#_Toc418971469)

[9.4 Részletes dinamikus nézetek 44](#_Toc418971470)

[9.5 Telepítési nézet és rendszerkörnyezet 48](#_Toc418971471)

[10 Az implementáció részletei 49](#_Toc418971472)

[11 Tesztelés 55](#_Toc418971473)

[11.1 Statikus tesztelés 55](#_Toc418971474)

[11.2 Dinamikus tesztelés 56](#_Toc418971475)

[11.2.1 Egység teszt 56](#_Toc418971476)

[11.2.2 Komponens teszt 57](#_Toc418971477)

[11.2.3 Integrációs teszt 58](#_Toc418971478)

[11.2.4 Rendszer teszt 59](#_Toc418971479)

[11.2.5 Átadási teszt 60](#_Toc418971480)

[12 Továbbfejlesztési lehetőségek 61](#_Toc418971481)

[13 A szakdolgozat tartalmi összefoglalója 62](#_Toc418971482)

[14 Irodalomjegyzék 64](#_Toc418971483)

[14.1 Felhasznált könyvek: 64](#_Toc418971484)

[14.2 Felhasznált honlapok 64](#_Toc418971485)

[15 Mellékletek 65](#_Toc418971486)

[15.1 Követelmény formátum sablonok 65](#_Toc418971487)

[15.1.1 Felhasználói követelmény sablon 65](#_Toc418971488)

[15.1.2 Rendszerkövetelmény sablon 66](#_Toc418971489)

[15.1.3 Formális használati eset szcenárió sablon 66](#_Toc418971490)

[15.1.4 User Story sablon 66](#_Toc418971491)

[15.1.5 Gherkin sablon 67](#_Toc418971492)

# Ábrajegyzék

# Célkitűzések

Dolgozatomban kitűzött célom egy olyan eszköz kifejlesztése, ami képes segíteni a szoftverfejlesztés egyes magas absztrakciós tevékenységeit. Gondolok itt főképp az eszköz felhasználói 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 közel valós időben láthassák egymás tevékenységeit a rendszert felhasználva. Végül, de nem utolsó sorban a rendszer legyen a lehető legkülönbözőbb környezetekből, 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. Végül 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 Ion Somerwille könyve [4] alapján készült.

### 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. Megfontolandó viszont az összekapcsolódó, de különböző csoportba sorolt követelmények gyors összevetésére is módot adni, de legalább egy hivatkozást elhelyezni a kapcsolódó követelményre. [4]

### Követelmények csoportosításai

Követelmények csoportosítása történhet a megfogalmazásuk mélysége szerint. 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. [4]

### 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. [4]

### 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 pontos és precíz megfogalmazása, mert a rendszerspecifikáció gyakran része a szerződésnek. A rendszerspecifikációból lehetetlen kizárni minden a tervezéshez köthető információt.

A rendszerkövetelmények megalapozzák a tervezést, illetve az implementációt, így a szoftver megalkotásában résztvevők a legfőbb olvasói. Ebből következik, hogy, 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. [4]

### 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. [4]

### 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. A teljes rendszerre vagy akár annak egy-egy összetevőjére is vonatkozhatnak. Amíg egy funkcionális követelmény nem megfelelő támogatása a rendszer csak kis részét érinti, addig, ha a rendszer nem teljesíti a nemfunkcionális követelményeket, az akár a teljes rendszert is használhatatlanná teheti. A nemfunkcionális követelmény nem csupán a rendszerre vonatkozhat, de a kifejlesztésének folyamatára is. Megszabhatja tehát a kifejlesztés közben használt metodikát, különböző minőségszabványok betartását írhatja elő, vagy akár megszabhatja a fejlesztés alatt használatos CASE eszközök sorát. Lehetséges, hogy egy adott nemfunkcionális követelmény megléte szükségessé teszi más, funkcionális követelmények 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. Például a gyors jelző kifejezhető inkább a másodpercenkénti tranzakciók számával vagy egy eseményre adott átlagos válaszidővel. [4]

### 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. Előírhatja, hogy az adott feladatot hogyan kell végrehajtani, hogy az 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. Az adott szakterület képviselőinek egyértelműek, azonban a rendszer tervezőinek némiképp idegenek lehetnek a nyelvezetük és a mögöttes implicit háttértudás hiányában. Ezt a problémát projectszótár használatával lehet némiképp orvosolni. [4]

### Szükséges és javasolt követelmények

A javaslat nem mindig követelmény, lehet, hogy csupán egy igény a rendszerre vonatkozólag, esetleg egy teoretikus elképzelés, ami jelenleg 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. [2] [6]

### Kulcsfigura

A kulcsfigurának nevezzük a rendszerrel kapcsolatba kerülő végfelhasználókat, és az összes olyan egyént, akire a szoftverrendszer bármilyen hatást gyakorol. Ilyen hatások lehetnek például a szoftver beüzemelése közben előforduló fennakadások hatásai, vagy akár a megrendelő cégen belüli személyek közti politikai erőviszonyok megváltozása, az újonnan bevezetett rendszer hatására. [4]

### 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. [4]

### 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. [6]

## 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 rendszerint külön tárolja a felhasználói és 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. [4]

### 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 gyorsabb, és olcsóbb 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, illetve mekkora annak az üzleti haszna. A megvalósíthatósági tanulmány információt biztosít a rendszer elkészítésének költséghatékonyságáról. A tanulmány elkészítésének végeztével döntés születik a fejlesztés folytatásáról. A részfolyamat kimenő dokumentuma a megvalósíthatósági jelentés. [4]

### Követelmények feltárása és elemzése

A folyamat során a potenciális felhasználókkal és megrendelőkkel történő megbeszélések, és egyeztetések során, illetve az esetlegesen 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 elősegíthetik 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.

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 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

A követelmény feltárás másik eszköze a forgatókönyvek vagy más szóval szcenáriók készítése, mely megkönnyítheti a kulcsfigurákkal való precíz kommunikációt, mivel azok ilyenkor a konkrét problémára, magára az üzleti folyamatra tudnak koncentrálni és azzal kapcsolatban véleményt formálni. A forgatókönyvek különböző eszközökkel és különböző formalitási szinteken készíthetők. [4]

### 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. [4]

### Szoftverkövetelmények dokumentuma

A szoftverkövetelmények dokumentuma, amit szoktak szoftverkövetelmény specifikációnak is hívni, az a dokumentum, mely a követelmény specifikáció folyamata során jön létre. A specifikáció 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. [4]

### Követelmények kezelése

A követelmények kezelése napjaink gyorsan változó világában igen fontos, ugyanis a rendszerrel kapcsolatos elvárások az idő előrehaladtával folyamatosan változhatnak. Megváltozhat 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. [4]

### 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. [4]

## A követelmények fontossága

A követelmények meghatározása, elemzése és karbantartása a megvalósítandó komplex szoftverrendszerek életciklusában egy kritikus fontosságú feladat, hisz ez a szakasz segít megérteni, 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. [4]

## 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. [4]

### 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 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 követelmény kulcsfontosságú részeit érdemes lehet kiemelni, félkövér, dőlt, vagy egyéb szövegformázási eszközökkel, ezzel elősegítve a követelmény későbbi olvasását, és a fontosabb információk hangsúlyozását. [4]

## 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, 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 szemléltetni, 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. [2]

## 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, szcenáriók 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 szcenárióval 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. [6]

### 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. Gyakran használják egy-egy használati eset kifejtésére. [6]

### 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 ilyen osztályokat a <<boundery>> sztereotípia jelzi. 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. [6]

### 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. Jól szemlélteti, hogy a modellezni kívánt rendszerviselkedésben a résztvevő objektumpéldányok mikor jönnek létre, mikor végeznek műveletet, milyen üzenetváltásokkal kommunikálnak egymással.

A diagram a feladatok sorrendjét, és időbeliségét nagyszerűen képes ábrázolni, de az elágazások, illetve a ciklikusságok jellemzésére, használható aldiagramok, OCL (Object Control Language) és megjegyzések átláthatatlanná és nehézkesen használhatóvá tehetik. Ezek szemléltetésére másik eszközt érdemes választani, például a tevékenység diagramokat.

A szekvencia diagramokon továbbá 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. [6]

### Együttműködési diagram

A szekvencia mellett egy másik a rendszerben megjelenő interakciókat mutató diagramtípus az együttműködési vagy más néven kommunikációs diagram. Itt viszont az időbeliség helyett hangsúlyosabb az objektumok szerveződése és a kapcsolataik. Az üzenetváltások hasonló célt szolgálnak, mint a szekvencia diagramoknál, de egymásutániságuk itt halványabban, egyszerű számozásként jelenik meg.

Amennyiben sok elem vesz részt az adott tevékenységben, de ezek között viszonylag kevés üzenetváltás zajlik, akkor e diagramtípus a szekvencia diagramnál praktikusabb szemléltető eszköznek bizonyulhat. [6]

### 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 is. [6]

### 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 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 értelmezett az úszósávok használata, amit többnyire a csomagok közti rétegződés (angolul layer) szemléltetésére használunk. [6]

## Az agilis követelménykezelés és modellezés

Az agilis követelményekkel kapcsolatos szemlélet szerint a hagyományos IEEE 830 szabványban lefektetett, és ahhoz hasonló megfontolások alapján megírt követelménydokumentumok egyik problémája, hogy 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. [8]

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 fednek 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.

Ezeket kiváltandó napjainkban egyre inkább terjednek el és veszik át a helyet az egyszerű szöveges User Story-k. [7][8][10][11]

### User Stories: A használati esetek agilis megközelítése

A User Story a használati esethez hasonlóan a felhasználó szemszögéből közelíti meg a rendszert, de nem egy folyamatot, hanem egy sokkal kisebb részt, a rendszer egy műveletét írja 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 projektben jelenlévő összes hátralevő feladatot fontosságuk sorrendjében tárolják. A kis méret további előnye, hogy a megváltoztatásuk és karbantartásuk is egyszerűbb. Amikor egy csapat az adott sztorit elkezdi elkészíteni, tehát amikor szükséges, természetesen a sztorit is részletezhetik.

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 és értéket képviseljenek. A méretük kezelhető legyen, tehát soha ne legyenek akkorák, hogy megnehezítsék a komplexitásuk és a fontosságuk megbecslését vagy a későbbi tervezésüket. [7][8][10][11]

## 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.

A szövegbányászattal kapcsolatos fejezetek megírásához Tikk Domonkos által szerkesztett Szövegbányászat című könyv [5] nyújtotta az alapot.

## 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. [5]

## 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. [5]

### 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. [5]

### 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.

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.

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. [5]

## 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. [5]

## 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. [6][9]

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. [4]

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. [6]

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.

Robert C. 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. [3]

## Az irodalomkutatás alapján leszűrt megfontolások

### A követelményelemzést támogató eszköz főbb tulajdonságai.

Az irodalomkutatásban megemlítettem számos eljárást a követelmények leírására, modellezésére. Az elkésztendő eszköz a követelmények felvitelére ezek közül, a napjainkban divatos Felhasználói történet (User Story) sablont nyújtja majd, melynek eredeti részei: a sztorit végrehajtó felhasználó, felhasználói cél, a cél eléréséből származó felhasználói üzleti haszon. Ezt kiegészítem egy kötelezően kitöltendő név mezővel, egy opcionálisan megadható leírás mezővel, és az analízist megkönnyítendő opcionális fontosság és komplexitás mezőkkel.

A követelménymodellezés támogatására az eszköz az UML használati eset diagramját biztosítja.

### A specifikációs szöveg és a specifikációs modell közti híd megteremtése

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 felhasználóinak, funkcióinak és osztályainak, illetve azok kapcsolatainak feltérképezésében. Ellenben ezek olyan szövegbányászati és szakirány specifikus eszközöket igényelnek, melyek túlmutatnak e dolgozat határain. Így az eszköz ezen a téren úgy kívánja segíteni a felhasználót, hogy a felhasználói történet sablonokban felvett új felhasználói szerepköröket, a modellezés résznél felkínálja majd használatra.

### 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.

Ezekből az elkészítendő szoftver arra vállalkozik, hogy minél inkább támogassa a kollaborációt és könnyű legyen használni. Ellenben nem célja, hogy a teljes fejlesztést végigkísérje se az, hogy más eszközökkel együtt működjön. Ezek reális továbbfejlesztési céloknak tekinthetőek.

# 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.

Ajánlott, 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

## Felhasznált könyvek:

[1] Eric E. Domain Driven Design Reference: Definitions and Pattern Summaries. *Dog Ear Publishing*, 2014

[2] Maksimchmuk, R. A., Naiburg, E. J.: UML földi halandóknak *Kiskapu Kft.*,2006

[3] Martin, R. C.: Tiszta kód. *Kiskapu Kft.*, 2010

[4] Sommerville, I.: Szoftverrendszerek fejlesztése. *Panem Kiadó Kft.*, 2007

[5] Tick D.: Szövegbányászat. *Typotex Kiadó Kft.*, 2007

[6] Vég, Cs.: Alkalmazásfejlesztés a Unified Modelling Language szabványos jelöléseivel. *Logos 2000 Bt.*, 1999

## Felhasznált honlapok

[7] Ambler, S. W.: UML 2 Use Case Diagrams: An Agile Introduction (<http://www.agilemodeling.com/artifacts/useCaseDiagram.htm>),   
utoljára megtekintve: 2014-12-11.

[8] Cohn, M.: Advantages of User Stories for Requirements (<http://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requirements>), utoljára megtekintve: 2014-12-11.

[9] Due, R. T.: Abbot Textual Analysis

(<http://www.informit.com/articles/article.aspx?p=29043&seqNum=9>),   
utoljára megtekintve: 2014-12-11.

[10] Ponomareff, D.: Agile stories, estimating and planning

(<http://www.slideshare.net/dimka5/agile-stories-estimating-and-planning>),   
utoljára megtekintve: 2014-12-11.

[11] Stellman, A.: Requirements 101: User Stories vs. Use Cases

(<http://www.stellman-greene.com/2009/05/03/requirements-101-user-stories-vs-use-cases/>),  
 utoljára megtekintve: 2014-12-11.

[12] Tablada, R.: We Are Complicating Things a Bit too Much (TDD & DDD)

(<http://ryantablada.com/post/we-are-complicating-things-a-bit-too-much-%28tdd-and-ddd%29>),   
utoljára megtekintve: 2014-12-11.

[13] Mészáros, M.: BDD demisztifikálva

(<https://www.youtube.com/watch?v=QJEJ406P5uw&index=29&list=PL4C41194CED00D8CD>), utoljára megtekintve: 2014-12-11.

[14] Szász Z.: Automata tesztelés és BDD

(<https://www.youtube.com/watch?v=j5DI8y7ueK0>),   
utoljára megtekintve: 2014-12-11.

[15] Wiki / Manual - Cucmber

(<https://github.com/cucumber/cucumber/wiki/Gherkin>),   
utoljára megtekintve: 2014-12-11.

# Mellékletek

## Követelmény formátum sablonok

### Felhasználói követelmény sablon

[SORSZÁM (pl:2.3.1)] [Követelmény neve - nagyobb és félkövér betűformátum].

[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 | …

…