**Dr. Sziray József – Kovács Katalin**

**AZ UML NYELV HASZNÁLATA**

****

Készült a HEFOP 3.3.1-P.-2004-09-0102/1.0 pályázat támogatásával.

Szerzők: Kovács Katalin

egyetemi tanársegéd

(3. fejezet, 4. fejezet, 7. fejezet, 9.1. alfejezet)

dr. Sziray József

egyetemi docens

(Bevezetés, 1. fejezet, 2. fejezet, 4.4. alfejezet, 5. fejezet, 6. fejezet, 7.7. alfejezet, 8. fejezet, 9. fejezet, 10. fejezet)

Lektor: dr. Kallós Gábor

egyetemi docens

© Szerzők, 2006

Az UML Nyelv használata A dokumentum használata A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 3 ►

A dokumentum használata

Mozgás a dokumentumban

A dokumentumban való mozgáshoz a Windows és az Adobe Reader meg szokott elemeit és módszereit használhatjuk.

Minden lap tetején és alján egy navigációs sor található, itt a megfelelő hivatkozásra kattintva ugorhatunk a használati útmutatóra, a tartalomjegy zékre, valamint a tárgymutatóra. A ◄ és a ► nyilakkal az előző és a követ kező oldalra léphetünk át, míg a Vissza mező az utoljára megnézett oldalra

visz vissza bennünket.

Pozícionálás a könyvjelzőablak segítségével A bal oldali könyvjelző ablakban tartalomjegyzékfa található, amelynek bejegyzéseire kattintva az adott fejezet/alfejezet első oldalára jutunk. Az aktuális pozíciónkat a tartalomjegyzékfában kiemelt bejegyzés mutatja.

A tartalomjegyzék használata

Ugrás megadott helyre a tartalomjegyzék segítségével Kattintsunk a tartalomjegyzék megfelelő pontjára, ezzel az adott fejezet első oldalára jutunk.

Keresés a szövegben

A dokumentumban való kereséshez használjuk megszokott módon a Szerkesztés menü Keresés parancsát. Az Adobe Reader az adott pozíció tól kezdve keres a szövegben.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 3 ►

Az UML Nyelv használata Tartalomjegyzék A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 4 ►

Tartalomjegyzék

**Bevezetés............................................................................................ 6 1. Objektumorientált szoftverek......................................................... 9** 1.1. A procedurális szoftverek jellemzői .......................................................... 9 1.2. Az objektumorientált szoftverek jellemzői ............................................11

**2. Egy objektumorientált fejlesztési modell..................................... 13** 2.1. Az UML nyelv megjelenése......................................................................13 2.2. A tervezési modell és a programkód viszonya.......................................15 2.3. A fejlesztési folyamat általános menete ..................................................17 2.4. A fejlesztés végigvitele...............................................................................24

**3. Use case modellezés..................................................................... 32** 3.1. A követelményelemzés szerepe a fejlesztésben.....................................34 3.2. A use case modell kialakítása....................................................................43 3.3. A use case modell dokumentálása ...........................................................67 3.4. Use case realizáció – a use case-ek megvalósítása.................................68

**4. Osztályok, objektumok, osztálydiagram...................................... 73** 4.1. Osztálydiagramok használata a fejlesztés különböző szakaszaiban.... 73 4.2. Objektum, osztály......................................................................................76 4.3. Speciális fogalmak, asszociációs viszonyok............................................88 4.4. A CRC-kártyák használata........................................................................97

**5. Interakciós diagramok ................................................................101** 5.1. Az objektumok közötti együttműködés ...............................................101 5.2. Szekvenciadiagramok ..............................................................................103 5.3. Konkurrens folyamatok és az aktiválások............................................106 5.4. Együttműködési diagramok....................................................................109 5.5. Felhasználási példa könyvtári kölcsönzésre .........................................111 5.6. A kétféle diagram összehasonlítása .......................................................114

**6. Csomagdiagramok ......................................................................116** 6.1. A szoftverműködés lebontása................................................................116 6.2. Függőségi viszonyok ...............................................................................116 6.3. Kibővítések a csomagdiagramban.........................................................120 6.4. Használati szempontok...........................................................................122

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 4 ►

Az UML Nyelv használata Tartalomjegyzék A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 5 ►

**7. Állapotmodellezés .......................................................................123** 7.1. Az állapotátmeneti diagram....................................................................123 7.2. Az objektum állapota ..............................................................................124 7.3. Az állapotátmenet ....................................................................................125 7.4. Állapothierarchia – Szuperállapot, szubállapotok ...............................135 7.5. Konkurrens állapotdiagram....................................................................136 7.6. Emlékező állapot......................................................................................138 7.7. Objektumok tesztelése állapotdiagram alapján....................................139

**8. Aktivitási diagramok ...................................................................141** 8.1. Folyamatok, tevékenységek leírása ........................................................141 8.2. Felbontási hierarchia ...............................................................................144 8.3. Aktivitási diagramok use case-ekhez.....................................................145 8.4. Úszópályák................................................................................................148 8.5. Egy tevékenység lebontása .....................................................................149 8.6. Használati célok .......................................................................................150

**9. Komponensdiagramok és telepítési diagramok .........................152** 9.1. A komponensdiagram.............................................................................152 9.2. A telepítési diagram összetétele .............................................................154 9.3. Felhasználási célok...................................................................................155

**10. Egy UML-modell összefüggősége és teljessége .......................157** 10.1. A modellezés általános problémái.......................................................157 10.2. Az összefüggőség (konzisztencia) .......................................................158 10.3. A teljesség ...............................................................................................160 10.4. Megállapítások........................................................................................161

*Felhasznált irodalom* ...........................................................................................162

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 5 ►

Az UML Nyelv használata Bevezetés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 6 ►

Bevezetés

A szoftvertechnológia területén az elmúlt tíz évben döntő mértékben meg-növekedett az objektumorientált fejlesztés súlya és jelentősége. Eb ben a folyamatban fontos szerepük volt azoknak a tervezési módszerek nek, programozási nyelveknek, ill. számítógépes segédeszközöknek, ame lyek a tíz év során jöttek létre és terjedtek el. Ide sorolható az a kiemelke

dő eredmény is, amely az UML nyelv megalkotásában nyilvánult meg. Az UML (Unified Modeling Language, jelentése magyarul: Egységes modellező nyelv) három kiváló szakember, Grady Booch, Ivar Jacobson és James Rumbaugh egyesített munkájának a terméke. A szerzők a nyelv legelső, 1.0-s változatát 1997-ben adták közre az Egyesült Államokban. Azóta az UML az objektumorientált irányzat egyik modern, széles körben felhasznált alapvető eszköze lett. Elterjedését nagymértékben előmozdítot ta, hogy grafikus elemeken alapszik, ami igen szemléletes és jól áttekinthe tő alkalmazhatóságra vezetett.

Az UML jelentőségét lényegesen növelte az a tény, hogy sikerült szab ványosítani. Megalkotói szerint „az UML egy általános célú vizuális mo dellező nyelv, amely arra használható, hogy specifikáljuk, szemléltessük, megtervezzük és dokumentáljuk egy szoftver rendszer architektúráját”. Az iménti meghatározással összhangban a nyelv grafikus formában teszi lehe tővé a szoftver specifikálását, ill. működésének modellezését, aminek alap ján a konkrét implementálást el lehet végezni. A kiinduláskor előállított ún. tervezési diagramok lényegesen különböznek az implementáláskor szöve ges formában előállított programozási forráskódtól. A két megjelenési forma ugyanannak a szoftvernek a definiálására szolgál, viszont ez a funk ció egymástól eltérő absztrakciós szinteken jut kifejezésre.

A tervezési folyamatban az UML-t a kiindulási fázisban használjuk, az zal a céllal, hogy minél biztonságosabban, áttekinthetőbben és megbízha tóbban készíthessük el a teljes szoftver tervét, az objektumorientált fej lesztési elvhez kapcsolódóan. Az így elkészült tervezési diagramok alapján

válik lehetővé a forráskód megírása és a futtatható szoftver elkészítése. Ebben a folyamatban természetes igény az, hogy az UML-fázis és a kódo lási fázis teljes összhangban legyenek egymással. Mint ismeretes, az ilyen jellegű ekvivalencia igazolásának folyamatát szoftververifikálásnak nevez zük.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 6 ►

Az UML Nyelv használata Bevezetés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 7 ►

Mindezek után megállapítható, hogy az UML nem más, mint egy ter vezési nyelv, ami egy szoftver rendszer minél megalapozottabb kidolgozá sának, elkészítésének folyamatát szolgálja. Ebben a tankönyvben bevezető áttekintést adunk az UML-ről, annak grafikus komponenseiről, az egyes

komponenseknek az objektumorientált fejlesztésben történő felhasználá sáról, valamint a nyelvnek a tervezési folyamatban betöltött modellezési szerepéről. Mindezzel a szoftvertervezés átgondolt és konzisztens végre hajtásának megalapozását kívánjuk elérni.

A könyvben nem szenteltünk teret egyik meglevő programozási nyelv felhasználásának ismertetésére sem. Ehelyett olyan általános elveket tár gyalunk, amelyek az implementációs nyelvek bármelyikénél alkalmazhatók. Az általános elvek megvalósítására azonban néhol bemutatunk egy-egy példát, elsősorban a Java nyelvre támaszkodva.

**A könyv tíz fejezetből áll, amelyek tartalma a következő:** Az 1. fejezet az objektumorientált szoftverek legfőbb jellegzetességeit foglalja össze, a klasszikus procedurális szoftverekkel történő összehason lításban.

A 2. fejezet bevezetést ad az UML nyelvről, majd itt kerül sor annak ismertetésére, hogy a grafikus tervezés és modellezés milyen kapcsolatban áll a forrásnyelvi megvalósítással. A további fejezetrészekben egy olyan fejlesztési modellt mutatunk be, ami szervesen illeszkedik az UML-alapú tervezéshez. Ebben a modellben a fejlesztési folyamat négy egymást köve

tő fázisra van felbontva.

Az UML több különböző modellezési, folyamatleírási lehetőséget biz tosít. Mindegyik modellezési módhoz külön grafikus megjelenítési forma, diagramfajta tartozik. A nyelvi elemeket, ill. használatukat a könyv 3. feje zetétől kezdve a 9. fejezetig terjedően mutatjuk be. Az egyes diagramfaj ták, bemutatásuk sorrendjében a következők:

• Use case diagram (használati eset diagramja): A szoftver rendszer kí vülről látható működését írja le, a felhasználók és a rendszerfunkciók egymással való kapcsolatának ábrázolásával. (3. fejezet.)

• Osztálydiagram: Az egyes osztályok belső felépítésének és az osztá lyok közötti statikus információs kapcsolatoknak a leírására szolgál. (4. fejezet.)

• Interakciós diagramok: Az egyes objektumok közötti együttműködést mutatják be, a köztük fennálló üzenetküldésekkel és azok hatásának

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 7 ►

Az UML Nyelv használata Bevezetés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 8 ►

feltüntetésével együtt. Két fajtájuk létezik: a szekvenciadiagram (ese ménykövetési diagram), ill. az együttműködési diagram. (5. fejezet.) • Csomagdiagram: Az egymással szorosabb funkcionális kapcsolatban álló osztályok csoportba sorolására, valamint az egyes csoportok kö zötti információs függőség leírására szolgál. (6. fejezet.)

• Állapotdiagram: Egy adott objektum különböző állapotait ábrázolja, azoknak a vezérlési feltételeknek a megadásával, amelyek az állapotvál tozásokat eredményezik. (7. fejezet.)

• Aktivitási diagram: A vezérlési struktúrákat, valamint az egyes tevé kenységek egymást követő, ill. egyidejű, párhuzamos lefolyását mutatja be. (8. fejezet.)

• Komponensdiagram, valamint a telepítési diagram: Az előbbi a szoft verkomponensek hierarchikus elrendezésének és a közöttük létesült kapcsolatoknak a leírására szolgál. Az utóbbi pedig azt ábrázolja, hogy egy szoftverrendszer komponensei milyen hardveregységeken helyez kednek el, és milyen kommunikációs kapcsolatok állnak fenn az egyes összetevők között. (9. fejezet.)

A tankönyv 10. fejezete azokkal a problémákkal foglakozik, amelyek egy adott szoftvertervhez tartozó különböző UML-diagramok egymással való összhangjára, konzisztenciájára vonatkoznak. Ugyanitt lesz még szó a di agramok teljességének feltételeiről is.

A fentiekhez még annyit fűzünk, hogy ezt a tankönyvet azoknak az egyetemi és főiskolai hallgatóknak ajánljuk, akik olyan informatikai kép zésben részesülnek, amibe beletartozik a szoftverfejlesztés is.

Budapest, Győr, 2006. május 15.

A szerzők

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 8 ►

Az UML Nyelv használata Objektumorientált szoftverek A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 9 ►

1. Objektumorientált szoftverek

1.1. A procedurális szoftverek jellemzői A klasszikus, hagyományos ún. *procedurális* elvű szoftverek már fél évszá zada vannak használatban, számos magas szintű programozási nyelv jött létre és terjedt el ebben a körben. Például: Algol, PL/1, Cobol, Fortran, Pascal, vagy a C. Ennek a területnek a modernebb ágát képviselik az ún. *strukturált* elvű szoftverek, amelyek jellemző nyelvei a Pascal és a C.

A procedurális programoknál a kialakított adatstruktúra szolgál arra, hogy a programegységek, programmodulok információt cseréljenek egy mással. Ezt a működési elvet az 1.1. ábrán szemléltetjük, ahol három mo dul, M1, M2 és M3 vesz részt a feladat megoldásában. A modulok rendel keznek saját, ún. *lokális adatokkal,* ezen kívül egymás között paraméterek kel is tudnak adatot váltani.

A procedurális programozási nyelveken megírt modulok, a belső ve zérlési feltételektől függően, egy-egy jól elkülönült utasítássorozat folya matos végrehajtásán keresztül fejtik ki működésüket. Ebben a megközelí tésben önálló modul lehet például egy Fortran-szubrutin, egy Pascal-pro cedúra, vagy egy C-függvény. A modulok által használt adathalmazok fel építésére nézve jellemző az azonos hosszúságú mezők ismétlődése, ill. a

fix hosszúságú rekordok használata.

Egy programrendszer működési folyamatait a különböző funkciókat megvalósító modulcsoportok együttese valósítja meg. A procedurális szoftvereknél a folyamatok és az adatok szét vannak választva. Ebből adódóan a funkciók megosztásán kívül még külön meg kell tervezni az adatstruktúrát is, mégpedig úgy, hogy az összhangban legyen a funkciók együttesével. Mint ismeretes, a megtervezett adatstruktúra összetétele, az adatok elrendezése igen nagy mértékben befolyásolja a számítási folyama

tok hatékonyságát.

A legújabb alkalmazások már egyedi, speciális felépítésű adatbázist igé nyelnek, amelynek szervesen kell igazodnia a feldolgozás jellegéhez. Ilyen alkalmazások, ill. adatbázisok a következők:

• *Számítógépes tervezőrendszerek (CAD: Computer-aided Design):* A mérnöki tervezési adatokat tartalmazzák, beleértve a tervezett komponenseket, a köztük levő kapcsolatokat, valamint a különböző tervezési verziókat.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 9 ►

Az UML Nyelv használata Objektumorientált szoftverek A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 10 ►

• *Számítógépes szoftvertechnológia (Computer-aided Software Engineering, CASE):* A szoftverfejlesztők számára szolgáló adatokat foglalja magában, mint például a forráskódot, a modulok közötti függőségeket, a felhasznált változók definícióját, a fejlesztési folyamat menetét.

• *Multimédia:* Képi, grafikus, térképészeti, hang, zenei, videó típusú in formációt kódoló adatokat tartalmaz.

• *Irodaautomatizálási rendszerek:* Dokumentumok tárolására, nyilvántartás ára, kezelésére, keresésére, ill. munkafolyamatok adminisztrálására szolgáló adatok találhatók bennük.

• *Szakértői rendszerek:* Nemcsak az adatokat, hanem az emberi szaktudást reprezentáló összefüggéseket, szabályokat is hordozzák.

M

Közös adatterület M

1 2

Saját adatok Hívás

Paraméterek

M

3

Saját adatok

Hívás

Saját adatok

Hívás

Paraméterek

**1.1. ábra.** Procedurális modulok együttműködése

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 10 ►

Az UML Nyelv használata Objektumorientált szoftverek A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 11 ►

1.2. Az objektumorientált szoftverek jellemzői Az *objektumorientált szoftverek,* röviden *OO-szoftverek* tervezési filozófiája, tervezési elve lényegesen eltér a procedurális elvű szoftverekétől. Az OO stratégia az információ elrejtésén alapul. Egy szoftver rendszert úgy kell tekintenünk, mint egymással kölcsönhatásban álló objektumokat, ame lyeknek saját, privát állapotuk van, ahelyett, hogy egy közös globális álla poton osztoznának. Ebben a megközelítésben a programkód és a hozzá tartozó adatok ugyanabban a szoftveregységben találhatók. Ezek az egysé gek maguk az objektumok.

Az OO-technológia alapvető koncepciója az, hogy a feladatok végre hajtása objektumok közötti *üzenetek küldése* következtében történik meg. Az ilyen jellegű működés megköveteli, hogy az objektumokhoz definiálva legyenek azok az üzenetek, amelyekre azok reagálnak.

Az OO-szoftverek fontosabb jellemző vonásai a következők:

• Az objektumok független egységek, amelyek önállóan működnek. Mindegyik objektum saját adatokkal rendelkezik, amelyek mindenkori aktuális értékei az objektum *állapotát* fejezik ki.

• Egy objektum működése a benne meglevő önállóan kezelt, elkülönített utasításcsoportok, az ún. *műveletek* (angolul *operations*), vagy más néven *metódusok (methods)* végrehajtása révén valósul meg.

• Az objektumok azáltal kommunikálnak egymással, hogy egymás műve leteit, metódusait hívják meg, ahelyett, hogy változóértékeket osztaná nak meg egymás között. Egy művelet hívása üzenetküldéssel történik meg. Az üzenetek ebben a szisztémában is hordozhatnak vezérlő pa ramétereket.

• Az objektumok által képviselt programok akár szekvenciálisan, akár párhuzamosan hajthatók végre. Párhuzamos végrehajtásnál az objek tumok műveletei egyidejű, konkurrens módon mennek végbe. Ez egy processzoros gépen nem jelent szó szerinti egyidejűséget, csak a vég rehajtás szervezésében, ütemezésében jelentkezik.

Az objektumok közötti együttműködés vázlatos sémáját az 1.2. ábrán lát hatjuk. Itt az O1, O2 és O3 objektumok szerepelnek együtt. Az OO-környezetben értelmezett üzenet nem jelent olyan fizikai jel küldést, mint ami a számítógép-hálózatok csomópontjai (hosztjai) között megy végbe. Esetünkben egy üzenet nem más, mint objektumok között továbbított, valamilyen konkrét feladatmegoldásra vonatkozó kérés.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 11 ►

Az UML Nyelv használata Objektumorientált szoftverek A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 12 ►

Üzenet

O

1

Műveletek (Metódusok)

Paraméterek

O

2

Műveletek (Metódusok)

Saját adatok

(Állapot)

Saját adatok Üzenet

Paraméterek

O

3

(Állapot)

Műveletek (Metódusok)

Saját adatok

**1.2. ábra.** Objektumok együttműködése

(Állapot)

Mivel egy objektum kizárólag üzeneteken keresztül tart fenn kapcsolatot a külső környezetével, ezért lehetőség van arra, hogy módosítsuk az objek tum változóit és metódusait, anélkül, hogy ez befolyásolná más objektu mok működését. Ez a lehetőség, amellyel élve egy objektumot úgy tudunk módosítani, hogy az nem hat ki a rendszer többi részére, az egyik legfon tosabb előnye az OO fejlesztési elvnek, szemben a procedurális fejlesztési elvvel. Az OO-szoftverek létrehozásához számos jól bevált programozási nyelv áll rendelkezésre. Ilyenek például: Smalltalk, C++, Java, Perl, PHP.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 12 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 13 ►

2. Egy objektumorientált fejlesztési modell

2.1. Az UML nyelv megjelenése

Az objektumorientált elemzési és tervezési módszerek az 1980-as évek végén, ill. az 1990-es évek elején jelentek meg. Ennek a fejlődésnek egyik fontos későbbi állomása volt az *UML nyelv* kidolgozása. Az elnevezés az angol *Unified Modeling Language* (magyarul: Egységes modellező nyelv) kez

dőbetűiből származik. A nyelv három kiváló szoftverfejlesztő, Grady Booch, James Rumbaugh és Ivar Jacobson együttes munkája révén jött létre, az Egyesült Államokban, 1997-ben. Ez volt az UML 1.0-s verziója. Azóta az UML az objektumorientált irányzat egyik modern, széles körben felhasznált alapvető eszköze lett. Elterjedését többek között annak kö

szönheti, hogy grafikus elemeken alapszik, ami igen szemléletes és jól átte kinthető alkalmazhatóságra vezetett. Az 1.0 után a szerzők elkészültek egy újabb változattal, a 2.0-val, amit 2004-ben bocsátottak ki.

A nyelv elterjedését nagymértékben előmozdította az, hogy sor került a szabványosítására. A szabványosítási folyamatot az OO-fejlesztés terüle tén kulcsszerepet játszó OMG elnevezésű (Object Management Group) konzorcium hajtotta végre. Az OMG konzorcium a Hewlett-Packard, az IBM, valamint a Microsoft cégekből tevődik ki, mindegyikük az informa tikai fejlesztések óriása az USA-ban.

Megalkotói szerint „az UML egy általános célú vizuális modellező nyelv, amely arra használható, hogy specifikáljuk, szemléltessük, megter vezzük és dokumentáljuk egy szoftver rendszer architektúráját”. Az iménti meghatározással összhangban a nyelv grafikus formában teszi lehetővé a szoftver specifikálását, ill. működésének modellezését, aminek alapján a konkrét implementálást el lehet végezni. A kiinduláskor előállított ún. *ter vezési diagramok* lényegesen különböznek az implementáláskor szöveges formában előállított programozási forráskódtól. A két megjelenési forma ugyanannak a szoftvernek a definiálására szolgál, viszont ez a funkció egymástól eltérő absztrakciós szinteken jut kifejezésre.

A tervezési folyamatban az UML-t a kiindulási fázisban használjuk, az zal a céllal, hogy minél biztonságosabban, áttekinthetőbben és megbízha tóbban készíthessük el a teljes szoftver tervét, az objektumorientált fej lesztési elvhez kapcsolódóan. Az így elkészült tervezési diagramok alapján

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 13 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 14 ►

válik lehetővé a forráskód megírása és a futtatható szoftver elkészítése, ami a fejlesztési folyamat végső célja. Ebben a folyamatban természetes igény az, hogy az UML-fázis és a kódolási fázis teljes összhangban legyen egymással. Mint ismeretes, az ilyen jellegű ekvivalencia igazolásának fo

lyamatát szoftververifikálásnak nevezzük.

Az UML arra szolgál, hogy vizuálisan specifikáljuk, megjelenítsük, ill. dokumentáljuk egy szoftverfejlesztés fázisainak eredményét. A nyelv igen hasznos a különböző tervezési alternatívák leírására, valamint az eredmé nyek dokumentálására. Az UML-diagramok egyaránt alkalmasak a megva lósítandó objektumorientált rendszer statikus és dinamikus megjelenítésére.

A statikus képet adnak: az osztálydiagram, a csomagdiagram, a telepí tési diagram, továbbá a komponensdiagram. Ezek az objektumorientált terv alkotó elemei között meglevő állandó kapcsolatokat dokumentálják.

A dinamikus képet a use case diagram, az aktivitási diagram, az inter akciós diagramok, valamint az állapotdiagram szolgáltatják. Az ide sorolt diagramok a működésben, vagyis a programfutás közben megnyilvánuló változásokat, „mozgásokat” tükrözik.

Az UML az OO-világban elterjedt Ada, Smalltalk, C++, vagy a Java nyelvek bármelyikéhez felhasználható a megírandó programok tervezésé ben. Abban a tekintetben sem korlátoz bennünket, hogy milyen jellegű szoftver elkészítéséhez használjuk. Egyaránt alkalmazható valós idejű, webes, hálózati, vagy akár adatfeldolgozó rendszerekhez is. Jóllehet ez egy grafikus nyelv, de mégis ugyanúgy rendelkezik szintaktikai szabályokkal, mint a karakterekre épülő nyelvek.

Önmagában véve az UML csak egy modellezési nyelv, semmilyen módszer, tervezési megfontolás nem képezi integráns részét. Az idevonat kozó tervezési módszerek ugyanakkor arra valók, hogy a nyelv felhaszná lására adjanak irányelveket, megoldásokat, jól beváló „recepteket”. Az ilyen jellegű módszerek harmonikusan összefüggő együttesét *módszertannak* vagy *metodológiának* nevezzük. Egyik ilyen fontos és széles körben elterjedt fejlesztési módszertan az, amelyet maguk a nyelvalkotók, Booch, Rum baugh és Jacobson dolgoztak ki és alkalmaztak számos UML-projektben. A módszertan elnevezése: *RUP (Rational Unified Process),* ami a tervezési lépések szigorú rendbe foglalására irányul.

Ennél a pontnál érdemes megemlíteni a következőket: A RUP elneve zésben a hangsúly a „Unified Process”-en van, ami „Egységes folyamat”-ot jelent. A „Rational” előtag annak a cégnek a nevéből származik, amelyben a három alkotó dogozott a módszertan közös megalkotásakor. Ez a cég a

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 14 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 15 ►

Rational, Inc. volt az USA-ban. A szoftvertechnológiában az UML-lel elért egyértelmű áttörés, és a cég által kifejlesztett Rational Rose elnevezésű CASE-eszköz piaci sikere után, 2003-ban az IBM felvásárolta a Rational, Inc.-et. Emiatt újabban már csak a Unified Process (UP) elnevezés kezdi jelölni a fejlesztési módszertant. A Unified Process jelenleg széles körben terjed a világon, a fejlesztési területek csaknem teljes spektrumát fedi le már.

2.2. A tervezési modell és a programkód viszonya Egy szoftver rendszer terve és a tervet realizáló forráskód között, leg alábbis elvben, szoros kapcsolatnak és konzisztenciának, összhangnak kell fennállnia. A terv, a kód és a kész rendszer közötti kapcsolatokat a 2.1. ábrán mutatjuk be.

Az ábrán különbséget teszünk a fordítási időben és a futási időben megjelenő reprezentációk között. A forráskód, ami például Java, vagy C++ nyelven íródott, olyan dokumentum, ami egyértelműen definiálja a program működését. Azt a működést, amit a lefordítás, betöltés és elindí

tás előz meg. Az UML diagram is egy dokumentum, ami egy rendszert specifikál, de ebből a megjelenési formából még nem lehetséges közvetle nül előállítani a forráskódot, mert nem létezik közöttük egy-egy értelmű megfelelés.

specifikál

**UML:**

UML modell Objektum-struktúra

absztrakt

nézetet ad

specifikál

absztrakt nézetet ad

**Programnyelv:**

Forráskód Működő program

Fordítási időben Futási időben **2.1. ábra.** A modell és a kód közötti kapcsolatok

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 15 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 16 ►

A tervezési diagramok és a forráskód egyaránt arra szolgálnak, hogy egy elvárt rendszert specifikáljanak, de abban különböznek egymástól, hogy ezt különböző absztrakciós szinteken fejezik ki. A forráskód a futtatható kód összes sajátságát kifejezi, hordozza, míg a tervezési modellből jórészt hiányzik a működésnek az a részletessége, ami a kódban megtalálható.

Az ábrán a vízszintes nyilak a közvetlen specifikációs kapcsolatot mu tatják, míg a függőleges nyilak az absztrakciós kapcsolatot. Ebben a felfogásban az UML tervezési modell a forráskódban levő in formáció absztrakciója. Mint ismeretes, az OO-programok működését maguk az objektumok valósítják meg. Ezek a futás közben létrejönnek, megszűnnek, miközben adatokat dolgoznak fel és kommunikálnak egy mással. Az erre vonatkozó modellösszetevő az ábrán az *objektumstruktúra.* Az objektumstruktúra annak az absztrakciója, hogy valójában mi történik akkor, amikor a program futásban, működésben van.

A fentiekből két következtetés vonható le:

• Először is az, hogy az UML nyelven megadott diagramok nem egysze rűen csak képek, hanem konkrét jelentésük van a tekintetben, hogy mit specifikálnak a rendszer működési sajátságaira vonatkozóan. Ebben a megközelítésben a diagramok olyanok, mint a programok, csak azok nál absztraktabb, elvontabb formában.

• Másodszor, az UML nyelv jelölésrendszere jól követi azt a szemlélet módot, amit a programozók alkalmaznak a kódírás folyamán. Ennek megfelelően, az UML és az OO-nyelvek ugyanazokkal a szemantikus alapokkal rendelkeznek, ami lehetővé teszi, hogy az UML-tervből kon zisztens programfejlesztést lehessen megvalósítani.

Az OO tervezési nyelvek és programozási nyelvek közös vonása, hogy egyaránt a szoftverműködés meghatározására szolgálnak. Az ún. *objektum modell* az a közös számítási modell, amit az UML és az OO programozási nyelvek osztanak meg egymással. A kétféle nyelv között szoros kapcsolat van, ahol is a nyelvek különböző absztrakciós szinteken fejeznek ki mű ködési vonásokat a programokról.

Az objektummodell nem egy specifikus modell az UML-ben, hanem inkább általános gondolkodási mód az OO-programok struktúrájáról. Olyan koncepciók kerete, amelyek arra használhatók, hogy megmagyaráz zák az összes tervezési és programozási tevékenységet. Amint a neve is kifejezi, az objektummodell alapja azoknak az objektumoknak az együtte se, amelyek a programot alkotják, amelyek a működését megvalósítják.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 16 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 17 ►

Ezekben testesülnek meg a számítógépi program alapvető vonásai, neve zetesen a bennük levő *adatok* és azok *feldolgozási folyamata.*

Egy adott objektum csak egy kis részét valósítja meg a teljes rendszer funkcionalitásának. A rendszer általános működése a különböző objektu mok közötti interakció révén valósul meg. Összegezve, az objektummodell a szoftverfejlesztésben azt fejezi ki, hogy úgy tekintjük az OO-programot mint egymással kommunikáló és együttműködő objektumok összességét.

Az objektummodell az alapja az UML tervezési modellnek. Az UML ben megvalósuló tervezési folyamat eredménye az egymással összekötte tésben levő és egymással kommunikáló *objektumok dinamikus hálózata.* Ezen belül a *statikus modellek* az objektumok között létező összeköttetéseket, azok topológiáját írják le. A *dinamikus modellek* az objektumok között kül dött üzeneteket írják le, valamint az üzenetek hatását az objektumokra*.* Mindezek az OO-programok futási, végrehajtási sajátságait is kifejezik. Mint látni fogjuk, a statikus és dinamikus modellt az UML ezekre a célok ra szolgáló diagramjai valósítják meg.

2.3. A fejlesztési folyamat általános menete 2.3.1. A fejlesztési fázisok

Az objektumorientált szoftverek fejlesztéséhez elterjedten alkalmazzák azt a négy fázisra, négy fejlesztési szakaszra bontott modellt, amelyet a 2.2 ábrán mutatunk be. A modell Booch, Rumbaugh és Jacobson munkájának az eredménye, az általuk kidolgozott RUP módszertan is ebből indul ki, és erre is épül. Az egyes fázisok, az eredeti angol elnevezésükkel együtt a következők:

• *Elindítás (Inception).*

• *Kidolgozás (Elaboration).*

• *Megépítés (Construction).*

• *Átmenet (Transition).*

(Construction)

(Inception) (Elaboration)

Megépítés (Transition)

Elindítás Kidolgozás 1 2 3 ... n Átmenet **2.2. ábra.** Az objektumorientált fejlesztés fázisai

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 17 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 18 ►

Az ábrán látható teljes folyamat igen fontos jellegzetessége, hogy egyrészt *iteratív,* másrészt pedig egyúttal *inkrementális* is. Mindez azt jelenti, hogy a szoftvert nem egyetlen kész adagban, egyszerre bocsátják ki a projekt vé gén, hanem e helyett egymás után következő külön fejlesztési darabokban, fejlesztési változatokban. Ekkor az egyes darabok, változatok fejlesztésé nek menete megismétlődik. Ezáltal a végső termék folytonos, több lépéses kidolgozású részek egymás után megépülő változataiként jön létre. Ezek a motívumok elsősorban a *megépítési* fázisban érvényesülnek, ahol számos ismétlődő, iteratív lépés után áll elő a szoftver. Egy-egy ismétlés, iteráció itt magában foglalja az összes szokásos életciklus-tevékenységet, ideértve az elemzést, tervezést, megvalósítást, valamint a tesztelést. A két motívum (iteráció és inkrementáció) egyébként a folyamat másik három fázisában is érvényre jut.

A végleges változat akkor jön létre, amikor már nincs szükség újabb ismétléses fejlesztés elvégzésére. Az *inkrementalitás* ebben a folyamatban úgy érvényesül, hogy mindegyik iterációban változik, vagyis módosul, bő vül, esetleg szűkül a termék. (A folyamat hasonlít például ahhoz, ahogyan Lev Tolsztoj a „Háború és béke” című regényét írta meg. A regényt az író nyolc alkalommal fogalmazta újra, átdolgozva azt elejétől végéig, amíg a maga számára is elfogadhatónak nem találta. Mindez tizenkét évet igé nyelt.)

Az *elindítási* fázisban az egész projekt üzleti értelmét határozzuk meg, valamint hogy mire terjedjen ki a projekt, mit foglaljon magában. Mindeh hez meg kell szerezni a projekt finanszírozójának a jóváhagyását, akinek el kell döntenie, hogy mekkora költséget tud vállalni.

A *kidolgozási* fázisban már részletesebben meghatározzuk a követelmé nyeket, magas hierarchikus szintű analízist és tervezést végzünk, azzal a céllal, hogy definiáljuk a szoftverarchitektúra alapjait, továbbá tervet ké szítsünk a *megépítés* fázisához.

Az *átmeneti* fázisra olyan teendők maradnak, mint például a bétateszte lés, a teljesítmény hangolása, vagy a felhasználók kiképzése. Még egyszer hangsúlyozzuk, hogy ebben a teljes folyamatban mind egyik fázis *iteratív* és *inkrementális* jellegű, több lépésre, ismétlésre és változ tatásra, módosításra alapozva. Mindez különböző mértékben érvényesül az egyes fázisokban, mégpedig leginkább a *megépítés* során.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 18 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 19 ►

2.3.2. A RUP-modell két dimenziója

Mint ismeretes, a szoftverfejlesztési életciklust leíró klasszikus vízesés modell öt alapvető fázisból áll:

1. Követelmények felállításának fázisa.

2. Elemzési fázis.

3. Tervezési fázis.

4. Megvalósítási fázis.

5. Tesztelési fázis.

Ez a modell egydimenziós, és egy fentről lefelé haladó tengelyen ábrázol ható. Az egyes fázisok között, a menet közben felmerülő módosítási igé nyek következtében, visszacsatolás jöhet létre.

A RUP-modell magában foglalja a vízesésmodellt is, ezáltal kétdimen zióssá válik. A két dimenzió oly módon jön létre, hogy a RUP négy fej lesztési fázisa a vízesésmodell öt fázisával van kombinálva. Ez úgy érten dő, hogy minden egyes fázis a vízesés-modell öt fázisa alatt képviselt *mun kafolyamatra (work flow)* van felbontva (2.3. ábra). Másként fogalmazva:

Mindegyik RUP fejlesztési fázisban öt egymást követő lépésben valósul meg a tervezési eredmény.

Átmeneti

fázis

Követelmények

felállításának

munkafolyamata

Elindítási fázis

Kidolgozási fázis

Megépítési fázis

Elemzési

munkafolyamat

Tervezési

munkafolyamat

Megvalósítási munkafolyamat

Tesztelési

munkafolyamat

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**2.3. ábra.** A kétdimenziós RUP-modell

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 19 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 20 ►

Az egyes fejlesztési szakaszok résztevékenységekre való bontása a teljes munkafeladat kisebb egységekre való szétválasztását teszi lehetővé. Mind ehhez kapcsolódik az iteratív és inkrementatív végrehajtási folyamat, ami a szoftverépítés egészét áttekinthetőbbé, hatékonyabbá és biztonságosabbá teszi.

A RUP-modellről megállapítható, hogy igen jól szolgálja a nagybonyo lultságú szoftverek létrehozását, mivel a feladat kisebb részekre való bon tását és a részek összehangolt, lépésenkénti kidolgozását teszi lehetővé. Amit még hozzátehetünk: A szoftvertechnológia jelenlegi fejlettségi szint jén, az objektumorientált szférában ez a modell alkalmazható a legelőnyö sebben a többi más modellhez képest.

2.3.3. Az egyes fázisok áttekintése

**Elindítás**

Többféle megvalósítása lehetséges, a projekt méretétől függően. Egy ki sebb projekt elindítható akár egy rövidebb megbeszéléssel is. Például ezzel a felvetéssel kezdve: „Tegyük ki a szolgáltatásaink listáját a webre.” Egy ilyen jellegű indításra elegendő lehet egy-két nap is. Ezzel szemben egy nagy projektnél elképzelhető, hogy egy részletes kivitelezhetőségi, megva lósíthatósági tanulmány elkészítésére van szükség. Ez a tanulmány akár

több hónapon keresztül is készülhet, több ember munkája révén. Az elindításnál igen fontos a projekt üzleti oldalának az alapos elemzé se. Ekkor meghatározandó, hogy egyrészt mekkora költségvonzata van a projektnek, másrészt pedig hogy mekkora bevétel, ill. haszon várható be lőle. A projekt méretének, költségeinek megállapításában döntő tényező a ráfordítandó emberhónap, valamint az átfutási idő. Mindehhez járul még a felhasználandó hardver és szoftver eszközök költsége is. Az így meghatá rozott költségeket, valamint az alapvető fejlesztési célokat részletesen egyeztetni kell a finanszírozóval. A finanszírozó egyaránt lehet az a cég, ahol a fejlesztés megvalósul, vagy pedig egy külső megbízó, amely meg rendeli a fejlesztést a maga számára.

Ebben a fázisban döntést lehet hozni arról is, hogy szükséges lesz-e megvizsgálni a projekt addigi menetét a következő, a kidolgozási fázisban, azzal a céllal, hogy módosítsanak a méretén, ill. a kitűzött célokon.

**Kidolgozás**

Az elindult projekt első teendői között át kell gondolni a követelményeket, jól meg kell érteni a problémát. Olyan kérdésekre kell választ adni, mint:

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 20 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 21 ►

Mi a cél? Milyen jellegű szoftverre lesz szükség? Milyen legyen a specifiká ciója, mit kell tudnia? Hogyan legyen az megvalósítva, hogyan fogjuk fel építeni? Milyen szoftvertechnológiát alkalmazzunk?

Ebben a fázisban szükség van a *kockázatok* előzetes átgondolására, elemzésére. A lehetséges kockázatok ugyanis olyan veszélyeket rejtenek, amelyek nagymértékben félrevihetik a projektet, főleg időben és költsé gekben, ami lehetőleg elkerülendő.

Martin Fowler, az „UML Distilled” című kiváló szakkönyvében (USA, 1997) a kockázatok négy kategóriáját különbözteti meg:

• *Követelmények kockázata:* A nagy veszély itt abban van, hogy olyan szoft vert tűzünk ki célul, amely az elkészülte után nem fog megfelelni a fel használónak, mert annak más lett volna a jó. A kidolgozás fázisában erre igen nagy gondot kell fordítani.

• *Technológiai kockázatok: Az itt felmerülő lehetséges kérdések például a következők:* – Milyen OO fejlesztési tapasztalattal rendelkeznek a fejlesztők? – Megoldható-e a feladat egy webkeresővel, ami egy adatbázishoz

kapcsolódik? Jó lesz-e ehhez a C++ nyelv? Vagy legyen a Java? • *Szaktudási kockázat:* Itt a fő kérdés az, hogy megvan-e a kellő létszám és szaktudás a kitűzött feladat megoldásához?

• *Vállalatpolitikai kockázat:* Azt kell itt elemezni, hogy a fejlesztő cég, vállalat gazdaságpolitikájába beleillik-e projekt. Vannak-e komoly erők a cégnél, akik támogatják, vagy pedig ellenzik azt.

Fowler szerint lehetnek még más egyéb kockázatok is, de azok nem min dig jelentkeznek. Ezzel szemben a fenti négy csaknem minden esetben megvan.

A kidolgozási fázis végső soron arra irányul, hogy a rendszer *architektú ráját* hozzuk létre. Ez az architektúra az alábbiakból tevődik össze:

• A követelmények részletes felsorolása.

• A feladat kidolgozásának modellje, ami már a szoftvert alkotó legfon tosabb OO-osztályokat is tartalmazza, felsorolva.

• A megvalósítás szoftvertechnológiája, annak részei, és illeszkedésük egymáshoz. Például, egy webes fejlesztés kliens-szerver kapcsolódása, az interfészek megvalósítási módja, a felhasznált programozási nyel vek, a fájlformátumok meghatározása stb.

• Az igénybe veendő hardver elemei, ill. a felhasználandó operációs rendszer.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 21 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 22 ►

Természetesen ez a fázis is egy iteratív folyamat, az architektúra is változá sokon eshet át. A változások szempontjából nem mindegy, hogy melyik programozási nyelvet használjuk. Például a Java elbírja a jelentős architek turális változtatásokat is. Ezt úgy kell érteni, hogy viszonylag kisebb plusz ráfordítással lehet követni a módosításokat. Ezzel szemben a C++ nyelv már nehezen viseli el az átalakításokat, használata minél stabilabb architek túrát tesz célszerűvé.

Fowler megfigyelései szerint a kidolgozási fázis a teljes projekt mintegy 20%-át viszi el időráfordításban. Ennél többet nem érdemes vele eltölteni. A véget érést a következő momentumok jelzik:

• A fejlesztők már megnyugodtak afelől, hogy jól meg tudják becsülni az emberhónapos ráfordítást, a főbb részfunkciók megvalósítására szét bontva.

• Mindegyik lényeges kockázat meg lett határozva, azonosítva, továbbá az is, hogy miként kell kezelni, leküzdeni ezeket a kockázatokat.

**Megépítés**

Ez a fázis tipikusan iterációk sorozatából áll, ahol mindegyik iterációs lé pés valójában egy mini projekt. A szoftver főbb részfunkcióira egyenként el kell végezni az alábbi tevékenységeket:

• elemzés,

• tervezés,

• kódolás,

• tesztelés,

• integrálás.

Az ismétlések sora akkor zárul le, amikor egy demóverzió adható át a fel használónak. Ezzel egyidejűleg sor kerülhet a rendszer szintű tesztelés végrehajtására.

A tesztelést és az integrálást nem szabad a fejlesztés végére hagyni. Minél később tesztelünk, annál nagyobb a veszélye annak, hogy sok hibát kell javítani a programokban, igen nagy ráfordítással. A késői tesztelés másik veszélye az, hogy ha a tesztek alapján módosítást kell végrehajtani a programon, az egy előrehaladott fejlesztésnél már költségesebb, mintha korábban oldottuk volna meg a módosításokat. Egyébként általánosan is igaz minden fejlesztésre, hogy minél később derül ki valamilyen tévedés vagy hiba, annál nehezebb a változtatást vagy javítást végrehajtani.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 22 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 23 ►

Fowler azt az elvet vallja, hogy a jó tesztelés megköveteli, hogy nagyjá ból ugyanannyi tesztkódot írjunk, mint amennyi programkódot magában a termékben. Ez az ökölszabály összhangban van azzal az általánosan elter jedt tapasztalattal, miszerint a tesztelés a fejlesztési folyamatban mintegy 50%-os arányban részesül.

Ennek a munkának a kódírással együtt kellene folynia. Mihelyt készen van a kód, elkészítendő a tesztprogramja is. Már a kódolás közben érde mes azon gondolkodni, hogy miként, milyen módon is fogjuk tesztelni az adott programunkat.

A részegységek, modulok tesztelését Fowler szerint maga a fejlesztő végezze, csakúgy, mint az integrációs tesztelést is. Ezzel szemben a teljes rendszer tesztelése már egy független, különálló team feladata kell legyen. Olyan teamé, amely kizárólag tesztelésre van szakosodva. Az ilyen mun kában részt vevők számára az a siker, ha minél több hibát tudnak felfe dezni, ami a szoftverminőség szempontjából mindenképpen kívánatos hozzáállás.

Mindehhez még annyit fűzünk, hogy az iteratív szakaszoknál is be kell tartani a részfeladatok teljesítésére megszabott határidőt. Ha ez nem sike rülne valamilyen oknál fogva, akkor újra kell egyeztetni a projekt menetét, a finanszírozót is bevonva.

**Átmenet**

Ez az utolsó fázis. Az a lényeges jellemzője, hogy itt már nem történik valódi, érdemi fejlesztés. A szoftver funkcionalitásán nem változtatunk már, és hibakeresés sem folyik ebben a fázisban. Ezek a tevékenységek lezárultak az előző fejlesztési lépcsőkben. Itt azokat a végső simításokat, javításokat, módosításokat hajtjuk végre, amelyek a szoftver optimálisabb működését eredményezik. Ez az optimalizálás a hatékonyabb, gyorsabb működés elősegítését szolgálja, másrészt pedig a felhasználói kényelem növelését.

Az átmeneti szakasz tehát arra irányul, hogy a nyers végtermékből egy optimalizált, „kicsiszolt” végtermék váljék. Tipikus példa lehet erre az az időszak, ami a béta kibocsátási verzió és a végső kibocsátási termék megje lenése között telik el.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 23 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 24 ►

2.4. A fejlesztés végigvitele

2.4.1. Kommunikációs problémák

A szoftverfejlesztésben az a legnagyobb kihívás, hogy egy jól működő rendszert hozzunk létre, olyat, ami kielégíti a felhasználói követelménye ket, éspedig ésszerű költségekkel megvalósítva. Ez egy valódi mérnöki feladat.

A feladat megoldását tovább nehezíti az, hogy a fejlesztők szóhaszná lata, zsargonja sok esetben lényegesen különbözik a felhasználók zsargon jától. Egy kórházi rendszer fejlesztésénél például meg kell küzdeni az or vosi kifejezésekkel, amik ráadásul még csak nem is magyarul vagy angolul

vannak. Ugyanilyen gondokkal járhat egy vasúti vezérléseket végző be ágyazott szoftver elkészítése is, amihez a vasutasok szakkifejezéseit kell megérteni.

A feladat pontos megértése és a félreértésektől mentes kommunikáció kulcsfontosságú a kifejlesztendő szoftver elfogadhatósága, minősége szempontjából. Ehhez annak a célnak az elérése kapcsolódik, hogy minél jobb, tökéletesebb specifikáció készüljön el, hiszen a specifikáción múlik az egész fejlesztés végeredménye. A tévedések csökkentésére az UML-ben egy alapvető fontosságú segédeszköz áll rendelkezésre: Ez a segédeszköz a *use case,* magyarul *használati eset.* (A tankönyvben általában megmaradunk az angol elnevezés használata mellett, mivel ez az elterjedtebb, és emellett könnyebb a kiejtése is.)

Egy-egy use case nem más, mint a rendszer kívülről látható viselkedé sének a leírása, definiálása. Ebben a megjelenítésben fel van tüntetve a felhasználó, valamint a számítógépes rendszernek azok az összetevői, amelyekkel a felhasználó együttműködésben, interakcióban van. Ez a kép felfogható úgy is, mint egy pillanatfelvétel a működés közben. Egy szoft vernek számos use case-e létezhet, attól függően, hogy milyen feladatokat, funkciókat lát el. A használati esetek összessége egy olyan külső képet ad a rendszerről, ami lehetővé teszi a teljes működés áttekintését a felhasználó oldaláról. Ugyanakkor mindez fontos támasz a fejlesztő számára is, aki ezáltal biztosabban megérti, hogy mit kíván a felhasználó.

A use case-ek központi szerepet játszanak az UML nyelv alkalmazása során. Nemcsak a megértést segítik elő, hanem hasznos eszköznek bizo nyulnak a projekt-tervezésben is, mivel ezekre lehet alapozni az iteratív fejlesztést. Ez azt jelenti, hogy a felhasználóhoz való visszacsatolásban és az ismétlődő módosításokban is fontos szerepük van, lehet rájuk támasz-

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 24 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 25 ►

kodni. Mindemellett nem csak a felszínen történő kommunikációhoz ad nak segítséget, hanem a belső, mélyebb fejlesztési folyamatokhoz is, ami vel a tankönyv későbbi részeiben foglalkozunk.

A használati esetek nem hordoznak információt a megvalósítás módjá ról, a rendszer struktúrájáról. A use case-eken alapuló megjelenítés végül is arra alkalmas, hogy a fejlesztőn kívül egyaránt el tudjon rajta igazodni a megrendelő, a felhasználó, a tesztelő, és egyáltalán mindazok, akik nem rendelkeznek ismeretekkel a szoftver konkrét felépítéséről, implementá ciójáról.

2.4.2. A kidolgozási fázis kockázatai

Ebben az alpontban részletesebben áttekintjük azokat a kockázatokat, amelyek a kidolgozási fázisban jelentkezhetnek. (Ezekről szó esett már korábban, az 1.2.2. alpontban.)

**Követelmények kockázata**

Ennek a kockázatnak a csökkentésében sokat tud segíteni az UML hasz nálata. Itt elsősorban a use case-ekre tudunk hagyatkozni, amelyek a teljes fejlesztési folyamat támaszai is egyben.

A use case-ek különböző terjedelműek, különböző bonyolultságúak lehetnek. A fő dolog az, hogy mindegyik egy funkciót jelöl ki, amit a fel használó is át tud látni, meg tud érteni, ami fontos is a számára. Ez a fajta segédeszköz hatékonyabbá és eredményesebbé teszi a finanszírozó és a fejlesztő közötti kommunikációt.

A kidolgozási fázisban meg kell találni az összes olyan use case-t, ami re a rendszer épülni fog. A gyakorlatban az nem lehetséges, hogy mind egyiket elő tudjuk itt állítani, de a legfontosabbakat, és az összes fontosat viszont meg kell találni. Ehhez arra van szükség, hogy interjúkat ütemez zünk be a felhasználóval, azzal a céllal, hogy use case-eket gyűjtsünk, amikhez szöveges kiegészítéseket is készítünk. Ezek nem kell, hogy túl részletesek legyenek, néhány bekezdésből álló rövid leírás elegendő hozzá juk. A szöveg a lényeget tartalmazó legyen, olyan, ami alkalmas arra, hogy a felhasználó egyértelműen meg tudja érteni, és láthassa belőle az alapel képzelést.

Az UML kidolgozói által javasolt módszertan nem azt igényli, hogy a teljes rendszert egy ütemben alakítsuk ki. Fontos, hogy meglegyenek a kulcsfontosságú szoftverosztályaink és use case-eink. A rendszert ezekre fogjuk alapozni, és további kibővítéseket tudunk majd hozzájuk építeni. A bővítések újabb use case-ek létrehozásával járnak (inkrementális fejlesz-

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 25 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 26 ►

tés), és ezek mind részei lesznek az iteratív folyamatnak is. A rendszer tehát lépésenként bővül, esetleges visszalépésekkel, javításokkal, módosítá sokkal együtt.

A use case-eken kívül itt jól felhasználhatók az UML más eszközei, nevezetesen az *osztálydiagramok (class diagram),* az *aktivitási diagramok (activity diagram),* a *szekvenciadiagramok (sequence diagram),* valamint az *állapotdiagramok (state diagram).* Ezek mindegyike a tervezési modell létrehozását szolgálja, miközben szoros kapcsolatban állnak a use case-ekkel és egymással is. (Ezekkel a könyv későbbi fejezeteiben részletesen fogunk foglalkozni.)

A diagramokat egy külön erre a célra létrehozott modelltervező team dolgozza ki. A team javasolt létszáma 2-4 fő. Annak érdekében, hogy ne tudjanak elveszni a részletekben, szoros határidőt kell nekik szabni.

**Technológiai kockázatok**

A technológiai kockázatok csökkentése érdekében ún. *prototípust* érdemes készíteni. A prototípus ebben az esetben egy megvalósított kísérleti prog ram, amely egy részfeladat megoldásának kipróbálására alkalmas. Ezzel azt lehet vizsgálni, elemezni, hogy egy kisebb szoftverrészlet milyen működést mutat.

Szemléltető példaként vegyünk egy olyan fejlesztést, amiben a C++ nyelvet szándékozzuk használni, egy relációs adatbázishoz kapcsolódva. A prototípus elkészítése a következő lépésekből állhat:

• Vesszük a C++ kompájlert és a hozzá tartozó fejlesztési segédeszkö zöket *(tool-okat).*

• A meglevő szoftvermodell (tervezési modell) valamelyik kiválasztott egyszerűbb részét leprogramozzuk. Eközben azt is nézzük, hogy mi ként tudunk boldogulni a felhasznált segédeszközökkel, azok hogyan funkcionálnak.

• Építsük fel az adatbázist, és kössük azt össze a C++ kóddal. • Próbáljunk ki több különböző segédeszközt. Tapasztaljuk ki, hogy melyikkel a kényelmesebb, hatékonyabb fejleszteni. Eközben szokjunk hozzá a tool-ok használatához, gyakoroljuk a lehetséges fogásokat.

Természetesen azt is szem előtt kell tartani, hogy a legnagyobb technoló giai kockázat nem magában egy szoftverkomponensben rejlik, hanem ab ban, hogy az egyes komponensek hogyan illeszkednek egymáshoz, miként működnek együtt. (Ez a tény persze igaz egy fejlesztő team tagjaira is, és a team egészére nézve is.) Az előző példára vonatkoztatva: Lehet jól ismerni

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 26 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 27 ►

a C++ nyelvet, lehet járatosnak lenni a relációs adatbázis kezelésében, de ettől még sok nehézséggel járhat a megírt program és az adatbázis összeil lesztése, együttes működtetése. Ezért is célszerű a fejlesztésben arra töre kedni, hogy a komponensek minél előbb össze legyenek illesztve, hogy minél előbb kipróbálhassuk az együttes működésüket. Mint tudjuk, ennek az az értelme, hogy a korábbi változtatások mindig kisebb veszteséggel járnak, mint a későbbiek.

Igen erősen szemmel kell tartani azokat a fejlesztési részterületeket, amelyeket később nehéz lesz megváltoztatni. Ugyanakkor törekedni kell a tervezés során arra is, hogy az egyes részelemeket minél könnyebben le hessen megváltoztatni.

Ehhez ilyen kérdéseket érdemes feltenni magunk számára:

• Mi lesz akkor, ha egy technológiai eszköz nem válik be? • Mi lesz akkor, ha két komponens nem illik össze?

• Mi a valószínűsége annak, hogy valami félresiklik a fejlesztésben, va lami baj lép fel? Mi legyen a megoldás, ha ilyesmi történik?

Ebben a folyamatban jól fel tudjuk használni a különböző UML-diagra mokat. Például, az osztálydiagramok és az *együttműködési diagramok (interac tion diagram)* igen hasznosak lehetnek abban, hogy megmutatják, hogyan kommunikálnak egymással a komponensek.

**Szaktudási kockázatok**

Sokan értékelik utólag úgy az OO-projektjüket, hogy az volt a gond, hogy kevés kiképzést kaptak a résztvevők. Sok helyen sajnálják a pénzt a kikép zésre, és utólag azért fizetnek többet, mert a projekt jóval tovább tartott

ahhoz képest, hogy nagyobb felkészültséggel vágtak volna bele. A kiképzés akkor ér valamit, ha az oktató a fejlesztésben járatos szak ember, aki el tudja mondani, hogy mire kell ügyelni, milyen hibákat kell elkerülni, és hogyan lehet bizonyos fontos feladatokat megoldani. (Az a jó, ha olyan tanítja, aki csinálja is.)

Fowler véleménye szerint az objektumorientált fejlesztést nem lehet tanfolyamokon, előadásokon megtanulni. A legjobb megoldás az, ha a tudást egy olyan személytől sajátítjuk el, aki úgy oktat, hogy közben maga is dolgozik a projektben, mégpedig huzamosabb ideig. Ezalatt megmutat

ja, miként kell csinálni egyes dolgokat, tippeket, tanácsokat ad, figyeli, ho gyan oldanak meg problémákat a résztvevők, és közben szükség szerint kisebb tanfolyamokat is tart.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 27 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 28 ►

Egy ilyen projektoktató személy *(mentor)* legalább az elején maga is tag ja a fejlesztő teamnek, és részt vesz a specifikáció kidolgozásában. Az idő előrehaladtával egyre inkább csak ellenőrzi, követi a fejlesztést, miközben maga egyre kevesebbet dolgozik a fejlesztésen. Legjobb az, ha a vége felé már feleslegessé is tud válni.

Érdemes a projektekhez ilyen külső szakértőket befogadni, és addig támaszkodni rájuk, amíg szükséges. Persze ez komoly költségnövelő té nyező, de a minőség érdekében sokszor megéri ezzel a lehetőséggel élni.

Az is megoldás lehet, ha csak arra alkalmaznak egy külső szakértőt, hogy időszakosan vizsgáljon át egy projektet (pl. havonta), és adjon véleményt róla, valamint tanácsokat a továbbfolytatásra. Ugyancsak elterjedt választás még az is, amikor a szakértő maga nem vesz részt a projektben semmilyen formában, viszont azt ellenőrzi, hogy minden az előírt fejlesztési technoló

gia szerint történik-e. Ezt *projekt-minőségbiztosításnak* nevezik. Több formája van: a folyamatos követéstől kezdve az időszakos ellenőrzésig. A szaktudási kockázatok csökkentésének egy másik természetes módja az önképzés. Fowler szerint érdemes kéthavonta elolvasni egy jó fejlesztési szakkönyvet, és ezt állandóan folytatni, annak érdekében, hogy valaki a szakmában tudjon maradni. A menedzsmentnek erre is pénzt kell áldozni, és a munkatársak számára időt biztosítani az önképzéshez.

**Vállalatpolitikai kockázatok**

Ez elsősorban gazdaságpolitikai, piaci jellegű terület, ahol a jelentkező kockázatok döntő mértékben nem műszaki természetűek. Ezen a téren minden a vállalti menedzsment üzletpolitikájától, üzleti stratégiájától függ. Az idevonatkozó problémák és megoldási módjuk kívül esnek ennek a tankönyvnek a keretein.

2.4.3. Az alaparchitektúra összetétele

A kidolgozási fázis eredménye abban áll, hogy létrehozzuk a szoftver *alap architektúráját.* Ez az architektúra a következő elemekből tevődik össze:

• A use case-ek halmaza, ami kifejezi a szoftverre vonatkozó követel ményeket.

• A tervezési modell, amely a szoftver működésének alapjait tükrözi, és a legfontosabb osztályok is leszármaztathatók belőle.

• A technológiai platform, amely megadja, hogy milyen szoftvertechno lógiai úton, milyen eszközökkel történik a megvalósítása a szoftvernek. Ide tartoznak a számítógépes eszközök, az operációs rendszer, a prog-

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 28 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 29 ►

ramozási nyelvek, fejlesztési környezet, tesztelési környezet, felhaszná landó tool-ok stb.

2.4.4. A megépítési fázis teendői

A kidolgozást követő *megépítési* fázisban iterációk, megoldási ismétlések sorozatára van szükség, amelyekben use case-ek vesznek részt. Ez egy tervkészítési folyamat, ami arra irányul, hogy mindegyik iterációhoz egy use case-t rendelünk. A tervkészítésben itt meg kell becsülnünk, hogy egy

egy use case-hez tartozó iterációs szakasz mennyi ideig fog tartani, hány emberhétből fog állni.

A becslésnek a következő tevékenységekre kell kiterjedni:

• elemzés,

• programtervezés,

• programkódolás,

• modultesztelés,

• modulok integrálása teszteléssel,

• dokumentálás.

A szoftverrendszer megépítése ebben a fázisban iterációk sorozatán ke resztül megy végbe, ahol mindegyik iteráció önmagában egy külön mini projekt. Ezek a projektek egyben inkrementálisak is, ami abban nyilvánul meg, hogy egy iteráció az őt megelőző iteráció use case-ére épül rá, azt bővíti, inkrementálja.

A programkódolás önmagában véve is iteratív folyamat. Az egyszer már megírt kódot, annak érdekében, hogy egyre jobb legyen a program, szükséges lehet újra írni, módosítani. A kód módosítása esetenként ko moly problémákkal járhat. Itt a következőkről van szó:

• Adva van egy program, amelyhez a fejlesztés alatt új funkciókat kell adni, mert ez vetődött fel. Ha egy eredetileg jól megtervezett és jól megírt kódhoz utólag teszünk újabb funkciót, akkor az eredeti kód „felborulhat”, az egész elveszítheti kiindulási rendezettségét. A prog

ramban nem volt benne az új funkció, nem volt beleolvasztva, ezért az új hozzáírás révén nem tud szervesen, összehangoltan illeszkedni a régi programhoz. Ez azért lehet így, mert az esetek többségében az új funkciót egyszerűen csak hozzáírják a régi programhoz, annak a tetejé

re vagy az aljára ültetik. Ez így nem igazán jó megoldás. A program túl bonyolult lesz, és nehézkesen fog működni.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 29 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 30 ►

• Elvileg egy teljesen jónak tűnő megoldás lenne az, ha a programot újraterveznénk, és ezután az új terv alapján újra megírnánk az egész kódot. Ez így rendben is volna, viszont túl nagy ráfordítással járna. Az új programban új problémák keletkeznének és új hibák lennének. Ugyanakkor azonban az is lehetséges, hogy az újratervezés hosszabb távon meg fogja érni, mert a végül összeállított kód egy kiegyensúlyo

zott, jól összehangolt működést fog eredményezni. A jól tervezett, át tekinthető programot tesztelni is könnyebb.

Mivel két irányban lehet választani, ebben a helyzetben egy mérnöki gazdasági döntés elé nézünk, ahol az optimális kompromisszumot kell megtalálnunk. Az egyik irány az, hogy újratervezzük, és úgy írjuk meg, a másik irány az, hogy a meglevőt írjuk át. A döntésben természetesen a kitűzött határidőt is szem előtt kell tartani. A szorító határidő természete sen az újratervezés ellen szól.

2.4.5. Az átrendezés elve

A meglevő kód átírására is dolgoztak ki olyan irányelveket, amelyek rende zett, áttekinthető módosításokat tudnak eredményezni. Az itt alkalmazott megoldást *átrendezésnek* (az angolban *refactoring*) nevezzük.

Az átrendezésben olyan technikákat, megoldásokat alkalmaznak, ame lyekkel nem változtatják meg a program funkcionalitását, hanem csak a belső struktúráját. A struktúraváltoztatással azt érjük el, hogy könnyebb lesz áttekinteni a programot, ezáltal könnyebb lesz tovább dolgozni vele.

Az átrendezési lépések apróbb változtatásokból állnak. Például: egy objektum metódusának átnevezése, egy adatmező átmozgatása egyik osz tályból a másikba, vagy két hasonló metódust egy közös osztályban össze vonva megjeleníteni. Jóllehet ezek önmagukban véve kis lépések, mégis azzal járnak, hogy nagyon sokat tudnak javítani a programon. Mindezek után már biztonságosabban és simábban lehet a programhoz hozzáillesz teni az új funkciókat.

Az átrendezés végrehajtását a következő elvek, fogások tudják meg könnyíteni:

• Szigorúan szét kell választani az átrendezést és a funkcióbővítést, va gyis a kettőt nem szabad ugyanabban az időben, együtt végezni. A két tevékenységet egymással váltogatva érdemes kivitelezni: átrendezés – funkcióbővítés, átrendezés – funkcióbővítés, …

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 30 ►

Az UML Nyelv használata Egy objektumorientált fejlesztési modell A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 31 ►

• Legyen készenlétben az átrendezés előtt a program tesztkészlete. A teszteket minél gyakrabban futtassuk le a módosítások során. Ezáltal hamar kiderül, hogy nem lett-e elrontva valami a belenyúlásokkal.

• Jól körülhatárolt, jól megfogható kis módosításokat hajtsunk végre, és mindegyik után rögtön teszteljünk is.

Mikor szükséges, ill. célszerű az átrendezés?

• Ha új funkciót adunk a programhoz, és nehézséget okoz a régi kódhoz igazodni. Ekkor a régi kódot át kell rendezni.

• Ha nehézséget okoz a régi kód megértése. Az átrendezés elvégzésével és a tesztelés végrehajtásával mélyebb betekintést nyerünk a kódba, ami a program további felhasználása és fejlesztése szempontjából lesz hasznos.

Előfordulhat, hogy olyan kódot kell átrendezni, amit nem mi írtunk, ha nem valaki más. Ilyenkor, ha még elérhető az eredeti szerző, akkor érde mes vele konzultálni arról, hogy miként is gondolkodott a kód megírása kor. A legjobb az, ha vele együtt, közösen nézzük át a kódot, és azután

látunk neki az átrendezésnek.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 31 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 32 ►

3. Use case modellezés

A valós üzleti folyamatok végrehajtását támogató szoftverrendszerek a folyamatok komplexitásából adódóan nagy bonyolultságú rendszerek. Fejlesztésüknél a megrendelő oldaláról, a felhasználói oldalról kell kiindul ni. A felhasználó az, aki részletesen ismeri az elvégzendő feladatokat, is meri a szervezet működését, azt a környezetet, amelyikben az alkalmazás működni fog. Ezen információk és ismeretek birtokában képes megfo galmazni azokat a célokat, amiket a szoftverrendszernek teljesíteni kell. Másrészt ő az, aki az alkalmazást működteti, használja.

Azokat a folyamatokat, amelyeket a különböző szervezetek valamilyen szoftveralkalmazással kívánnak támogatni, a RUP terminológia üzleti fo lyamatként definiálja. A számítógépes támogatást igénylő üzleti folyama tok leírására, az üzleti folyamatokat érintő elemek feltárására a RUP mód szertan *üzleti modellezés (Business Modeling)* szakaszában kerül sor.

Az üzleti modellezés a rendszerfejlesztési folyamatnak az a szakasza, amelyik még nem a szoftverfejlesztésre koncentrál. Az üzleti modellezés során felállított modellek célja, hogy a felhasználók és a fejlesztők számára egységes kép alakuljon ki a szervezetről, a szervezetben zajló folyamatok

ról, arról a környezetről, amelyikben a szoftverrendszer működni fog. En nek a munkaszakasznak az eredményeként elkészül az a részletes üzleti dokumentum, amelyik összefoglalja a felhasználó igényeit, elképzeléseit, pontosan leírja a szervezetben lejátszódó folyamatokat, amelyek számító gépes támogatást igényelnek. A rendszerfejlesztés során ugyanis nemcsak szoftverrendszert tervezünk, a fejlesztés teljes folyamatába beletartozik az a szervezet, amely számára a rendszert fejlesztjük, és azoknak az üzleti folyamatoknak a vizsgálata, amelyek számítógépes támogatását akarjuk megvalósítani. Az üzleti dokumentum részletezi a fejlesztés szempontjából releváns üzleti folyamatokat, a folyamatok más folyamatokkal való kapcso lódását, pontosan leírja a folyamatok során elvégzendő tevékenységeket, meghatározza a folyamatok bemeneteit és kimeneteit, ismerteti a folyama

tokban érintett szereplők (üzleti aktorok, üzleti entitások stb.) körét. A rendszerfejlesztési folyamat során a tényleges szoftverfejlesztési munka a követelményspecifikáció feladataival kezdődik. Ebben a munka szakaszban az *MDA (Model Driven Architecture – Modell-vezérelt Architektúra)* szabvány által definiált, a fejlesztendő rendszer struktúráját leíró modellek közül a *PIM (Platform Independent Model – Platformfüggetlen Modell)* modell kialakításához kezdünk hozzá.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 32 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 33 ►

A tankönyvben a szoftverfejlesztési lépések közül a követelményspeci fikáció során végrehajtandó tevékenységeket konkrét gyakorlati példákon keresztül szemléltetjük. A példa teljes mélységű kidolgozására a tankönyv keretein belül nincs lehetőség, azonban a feladatok kidolgozásakor próbál tunk arra koncentrálni, hogy a követelményspecifikáció munkafolyamat, a munkafolyamat során végrehajtandó tevékenységek, a végrehajtás mene tének logikája egyértelmű és könnyen követhető legyen. A use case model lezés fejezetben a mintapéldához kapcsolódó szövegrészek kezdetét és végét egy vízszintes vonal jelzi, a leírásokat, magyarázó szövegeket a nor mál szövegtől való megkülönböztetés érdekében dőlt betűvel írjuk.

**Mintapélda**1

A SWEProducts cég különböző termékek, áruk forgalmazásával foglalko zik. A cég a termékek értékesítése mellett további szolgáltatásokkal is ren delkezésre áll ügyfelei részére. Árajánlatot készít, vásárlás esetén biztosítja a termékek kiszállítását, átvállalva a kiszállítás költségeit. A cég ügyfélcent rikus szellemben működik, amit mi sem bizonyít jobban, mint az Interne ten is elérhető szolgáltatása, az Internetes árajánlat készítésének lehetősé ge. Az Internetes szolgáltatás kiterjesztésével olyan ügyfeleket céloznak meg, akik pontosan meg tudják adni a termékek megvásárlásához szüksé ges adatokat. Interneten keresztül történő árajánlat szolgáltatás igénybe vétele esetén pontosan ismerni kell a termékek paramétereit (pl. típus), és pontosan meg kell tudni adni a kiválasztott termékből kívánt mennyiséget (darabszámot).

Az árajánlatokkal, a megrendelések kezesével kapcsolatos feladatokat a cég ügyfélszolgálati munkatársai látják el.

A cégnek az árajánlatok összeállítása, az értékesítés mellett el kell látni a termékbeszerzéssel és a kiszállítással kapcsolatos feladatokat.

1 A mintapélda kidolgozásakor nem határoztuk meg, hogy a cég konkrétan milyen terüle ten folytatja tevékenységét. Azoknak viszont, akik a gyakorlat példáján könnyebben értik meg az elméleti ismereteket, javasoljuk, gondoljanak egy olyan cégre, amelyik különféle alkatrészek forgalmazásával foglalkozik.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 33 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 34 ►

<<business subsystem>> Árajánlatkészítés és megrendeléskezelés

<<business subsystem>> Beszállítói megrendelések

<<business subsystem>> ÁruKiszállítások kezelése

<<business subsystem>> Pénzügyi tranzakciók

Üzleti folyamatokat leíró csomagok

és a közöttük fennálló függőségi kapcsolatok – részlet

Az üzleti modellezés során összegyűjtött információk rögzítésére is hasz nálhatjuk az UML modelleket. A fenti ábrán UML csomagokat láthatunk, amelyek a cég által végzett üzleti folyamatokat modellezik. A *csomag* az az eszköz az UML-ben, amibe a logikailag összetartozó modellelemeket fog hatjuk össze, különíthetjük el. A folyamatok közötti kapcsolatot függőségi viszonyt2 (dependency) reprezentáló szaggatott nyilak jelölik.



3.1. A követelményelemzés szerepe a fejlesztésben A szoftverfejlesztési folyamat első lépéseként azt kell meghatározni, mi a szoftverfejlesztés célja. A 3.1. ábra azt hivatott szemléltetni, hogy a szoft verfejlesztési tevékenység célja nem más, mint a követelményeknek, a kor látozásoknak, a megszorításoknak (anyagi, erőforrásbeli, emberi stb.) meg felelő szoftverrendszer megtervezése, megvalósítása, üzembe helyezése.

Követelmények 

Korlátozások, 

**Üzleti**

**dokumentáció**

megszorítások **Fejlesztés során elvégzendõ feladat**

**Felhasználói igényeket kielégítõ**

**szoftverrendszer**

**3.1. ábra.** A szoftverfejlesztés összetevői

2 A függőségi viszonyt a 4.3.7. alpont ismerteti.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 34 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 35 ►

A kész szoftvertermék kifejlesztéséhez szükséges feladatok végrehajtásá hoz, pontos specifikálásához a felhasználó igényeit és az üzleti folyamatok működését összefoglaló üzleti dokumentumból kell kiindulni. A részletes üzleti dokumentum alapján kell meghatározni:

• A megvalósítandó szoftverfunkciók halmazát, céljait és indokait; • A tervezett rendszer működésére vonatkozó előre látható korlátozáso kat, amelyeket például a megbízó pénzügyi háttere, pillanatnyi elképze lései támasztanak;

• Azokat a feltételeket, korlátozásokat, amelyeknek meg kell felelnie a fejlesztendő rendszernek ahhoz, hogy késznek mondhassuk és átad hassuk.

A fejlesztés kezdetén továbbá egyeztetni kell a szakkifejezések használatát a félreértések elkerülése érdekében.

3.1.1. Felhasználói követelmények

A felhasználónak a szoftverrendszerrel szembeni igényei, elvárásai, azaz a felhasználói célok ún. felhasználói követelmények formájában fogalma zódnak meg. A *felhasználói követelmények* a tervezett szoftverrendszer külső környezetét, viselkedését a felhasználó fogalmaival írják le. A felhasználói követelmények leírásának különböző módszerei ismertek. A leírást készít hetjük természetes nyelven, intuitív diagramok rajzolásával, használhatók elterjedt ábrázolási technikák pl. folyamatábrák, készíthetünk áttekintő, a megértést segítő táblázatokat.

A felhasználók, mint megrendelők általában a szoftverrendszernek csak az alkalmazás felhasználói oldali képét (felületét), nézetét tudják jól leírni. Meg tudják adni, le tudják írni a szervezetnek, ill. a szervezet műkö désének azt a területét/területeit (alrendszer), azt a szervezeti környezetet, amelyikben majd a fejlesztendő szoftveralkalmazás működni fog. Megad ják a rendszertől elvárt szolgáltatásokat (szoftverfunkciók, amiket a szoft ver végrehajt), és azokat a feltételeket (megszorításokat), amelyeket a rend szer fejlesztése és majdani működése során be kell tartani. Vannak elkép zeléseik az alkalmazás használatáról, az alkalmazás bemeneteiről és a tőle elvárt kimenetekről (pl. listák, bizonylatok formája). A fejlesztő feladata lesz, hogy a rendszer viselkedését ezen információk alapján pontosan de finiálja, modellezze.

A rendszer modellezéséhez, leírásához a fejlesztőnek több feladatot kell megoldania. A felhasználói követelményeket (ún. magas szintű célok)

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 35 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 36 ►

kategorizálni kell3, közöttük prioritási sorrendet kell kialakítani, majd a felállított fontossági sorrend alapján szükséges mélységben részletesen ki kell dolgozni azokat. Csak a pontosan ismert és részletezett felhasználói célok ismeretében van lehetőség a szoftverrendszertől elvárt konkrét szol

gáltatások meghatározására. A szoftverfejlesztésnek ezen szakaszát köve telményelemzésnek (követelményspecifikációnak) nevezzük. Az alkalmazástól elvárt szolgáltatások, szoftverfunkciók a követel ményspecifikációban *funkcionális* és *nem funkcionális követelmények* formájá ban realizálódnak.

3.1.2. Funkcionális követelmények

A *funkcionális követelmények* a szoftverrendszer működésére, a tervezett rendszer szolgáltatásaira, a tényleges funkcionalitásra vonatkozó leírások, amik a szoftverrendszert kívülről nézve, a felhasználó szemszögéből írják le. Legtöbb esetben nem a felhasználó valódi céljait (magas szintű felhasz

nálói célok – felhasználói követelmények) fogalmazzák meg, hanem pon tosan azokat a szolgáltatásokat, amiket a felhasználó igénybe kíván venni, azokat a tevékenységeket (funkciók), amiket a felhasználó a rendszerrel el akar végezni, végre kíván hajtani.

A funkcionális követelmények továbbá leírják, hogy a rendszert ért ha tásokra, eseményekre a szoftveralkalmazásnak hogyan kell reagálni, a meg fogalmazott feltételek, a megadott megszorítások függvényében a rend szernek milyen alternatív végrehajtást kell biztosítani, a bekövetkezett kül ső események hatására milyen más funkciókat kell aktivizálni. A fejlesztés nek ebben a szakaszában a szoftverrendszer belső struktúráját, belső mű ködését ún. „fekete doboz”-nak tekintjük. Csak arra koncentrálunk, hogy

meghúzzuk a fejlesztendő rendszer határát, pontosan definiáljuk, hogy a fejlesztendő szoftverrendszernek milyen funkciókat kell megvalósítani, ill. mely funkciókat nem kell támogatni.

A felhasználói célokat a rendszer a követelményspecifikációban defini ált szoftverfunkciók megvalósításával fogja teljesíteni. A szoftverfunkciók leírását, a rájuk vonatkozó funkcionális követelményeket UML modellezés esetén *use case*-ek formájában modellezzük, írjuk le. A RUP modelljének filozófiáját követve a részletes kidolgozási fázis végére minden felhaszná-

3 A követelmények kategorizálásnak és minősítésének számos hatékony módszere létezik. A szakirodalom a Software Engineering Institute (SEI) és az ISO által kidolgozott mód szereket javasolja alkalmazni. Az egységesített módszertan a követelmények csoportosítá sát a FURPS+ modell alapján végzi.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 36 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 37 ►

lói célhoz tartozni kell a funkcionális követelmények egy halmazának, de legalább egy use case-nek.

A fejlesztés során vannak szituációk, amikor a felhasználói célok és a funkcionális követelmények nem határolhatók el tisztán, egzakt módon. Habár az UML nem tartalmaz eszközt a felhasználói célok és a funkcioná lis követelmények konzekvens megkülönböztetésére, kezelésére, a két fogalom egyedi értelmezésére vonatkozóan, a fejlesztés hatékony menete érdekében viszont, azokat szükséges tudatosan külön kezelni. A felhaszná lói célok és a funkcionális követelmények elhatárolásának érdekében az alábbi pontokat érdemes figyelembe venni:

• elsőként a felhasználói célokat definiáljuk, majd sorra kell venni azo kat a use case-eket, amelyek azokat teljesítik,

• ahol a funkcionális követelmények és a felhasználói célok nem azonos értelmezést kapnak, azokat tudatosan elkülönítve kezeljük, • a felhasználói célok az alternatív utak átgondolására adnak lehetőséget, amellyel a felhasználói célok teljesíthetők,

• a funkcionális követelményeket elemzési, tervezési célokra használjuk, • a részletes kidolgozási fázis végére minden felhasználói célhoz tartoz ni kell a funkcionális követelmények (use case-ek) egy bizonyos kész letének.

A működésnek azonban vannak korlátai, feltételei, megszorítások, ame lyeket a tervezés során szintén figyelembe kell venni. A működést befolyá soló elemek halmazát *nem funkcionális követelményeknek* nevezzük.

3.1.3. Nem funkcionális követelmények

A felhasználói célok összegyűjtése során találkozunk olyan megfogalmazá sokkal, amelyek nem konkrétan a szoftverrendszer működésére, a szoft verrendszer által megvalósítandó szolgáltatásokra irányulnak, hanem a szoftverrendszer működését befolyásolják. Ilyenek a működés környezeté re vonatkozó kényszerek, feltételek. Ezeket a követelményeket nem funk cionális követelményeknek nevezzük.

A nem funkcionális követelmények vonatkozhatnak a rendszer egészé re, vagy egy részére, vagy akár egy konkrét funkcionális követelményre. A nem funkcionális követelmények halmazába tartoznak olyan megkötések, feltételek, megszorítások, mint például:

• a szoftverrendszer hardver- és szoftverkörnyezete,

• a rendszer válaszideje,

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 37 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 38 ►

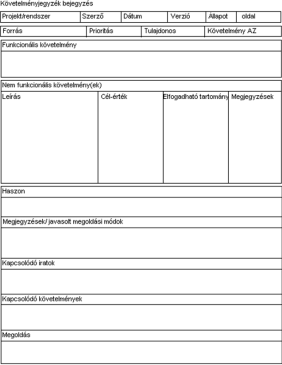
• a rendszer megbízhatósága,

• biztonsági szempontból a mentések gyakorisága,

• rendszerösszeomlással kapcsolatos követelmények,

• a szoftverrendszer fejlesztésére vonatkozóan technológiai, technikai és egyéb előírások (a fejlesztési folyamat modelljeinek leírására az UML nyelvet kell használni) stb.

A funkcionális és nem funkcionális követelmények meghatározáshoz és dokumentálásához használhatjuk a *„Követelményjegyzék-bejegyzés” formalapot.* A 3.2. ábrában bemutatott minta-formalap segít a követelmények ponto

**3.2. ábra.** Követelményjegyzék-bejegyzés formalap

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 38 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 39 ►

sabb, precízebb, ezáltal ellenőrizhetőbb követelményekké való formálásá ban. A fejlesztési munka korai szakaszában nem biztos, hogy a formalap minden részét ki tudjuk tölteni, a fejlesztés elején lesznek hiányosságok. A fejlesztés előrehaladtával, egyre több információ birtokában a követelmé nyekhez készített formalapot pontosítjuk, szükség esetén módosítjuk egé szen addig a pontig, amikor a felhasználók és elemzők egyetértésre nem jutnak abban, hogy a formalap már teljes leírását adja a leendő rendszernek.

„Követelményjegyzék bejegyzés” formalap kitöltését 3.1. táblázatban összefoglalt kitöltési útmutató segíti. A segédlet részletes leírást, magyará zatot ad a formalapon szereplő minden mezőre.

**3.1. táblázat.** Kitöltési útmutató

| **Követelmény AZ** | egyedi azonosító |
| --- | --- |
| **Forrás** | a követelmény forrása, lehet személy, dokumentum stb. |
| **Prioritás** | a követelmény prioritása, a felhasználó szerint, pl. ma gas/alacsony, vagy kötelező/ javasolt/ választható |
| **Tulajdonos** | felhasználó vagy felhasználói szervezet, aki a követel ménnyel kapcsolatos egyezkedésért felelős |
| **Funkcionális köve telmény** | az igényelt lehetőség vagy szolgáltatás leírása |
| **Nem funkcionális követelmény** | leírás, lehetőség szerint cél értékkel, elfogadható tarto mánnyal (minimum, maximum), minősítő megjegyzéssel |
| **Haszon:** | a követelmény kielégítéséből származó várható hasznok leírása |
| **Megjegyzések/ java solt megoldási módok** | lehetséges megoldások leírása, általános megjegyzések |
| **Kapcsolódó iratok** | hivatkozás kapcsolódó dokumentumokra, mint például felhasználói dokumentumok, projektalapító okirat, adat folyamábra stb. |
| **Kapcsolódó követel mények** | ha különböző követelmények hatnak egymásra, vagy kizárják egymást, akkor a hivatkozást fel kell jegyezni mindkét oldalon, hogy esetleges változtatás esetén fel lehessen mérni a hatást a mások oldalon |
| **Megoldás** | a követelmény megoldási módjának feljegyzése, például egy konkrét funkcióleírásra való hivatkozással. Ha egy követelményt nem fogunk kielégíteni, akkor itt kell felírni az okait. |

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 39 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 40 ►

3.1.4. Szakterületi követelmények

A követelmények csoportosításakor, strukturálásakor találkozunk olyan követelményekkel, amelyek nem közvetlenül a felhasználói igényekből származtatva a tervezett szoftverrendszer működésére (funkcionalitására), ill. a működés környezetére vonatkoznak, hanem a leendő rendszer által kiszolgált szakterület követelményeiből adódnak. Ezek a követelmények a tervezett rendszer szakterületén alkalmazott előírásokat, megszorításokat (pl. számítási előírások, az adott szakterület működésére vonatkozó jog

szabályok stb.) foglalják össze. A követelményeknek ezt a csoportját *szak területi követelményeknek (Domain Requirements)* hívjuk.

A szakterületi követelményeket összefoglaló dokumentum tartalmaz hat olyan, a szakterület sajátosságaiból adódó előírásokat, amelyek:

• újabb funkcionális és nem funkcionális követelményeket generálnak, vagy

• a már megfogalmazott követelményekre korlátozásokat írnak elő, de előfordulhat az is, hogy

• pontosan meghatározzák egy szoftverfunkció konkrét végrehajtási módját.

A szakterületi követelmények – az alkalmazási terület sajátosságait erősen magukon hordozva – sok esetben hiányosak. A hiányosság fő oka, hogy a szakterület képviselői számára a szakterületen alkalmazott szabályok, elő írások olyannyira nyilvánvalóak és egyértelműek (implicitás), hogy fel sem merül a szándék azoknak a fejlesztés során történő megemlítésére, így azok pontos értelmezésére, magyarázatára sem kerül sor. Mindehhez hoz zátársul az is, hogy a szakmaterület által használt szaknyelv (szakzsargon) a fejlesztők számára nehezen érthető. Az implicitás és a két szakma (fejlesz tő–felhasználó) szaknyelvének különbözősége sok esetben végzetes félre értésekhez vezethet, aminek az eredményeként olyan termék kerülhet ki fejlesztésre, ami nem az eredeti céloknak megfelelően fog működni. A fentiek elkerülése érdekében a szakterületre vonatkozó sajátosságokat, előírásokat, szabályokat a fejlesztési munka során megfelelő hangsúllyal kell kezelni, a szakmaspecifikus kifejezéseket pedig – erre a célra rendsze resített – külön fogalomszótárban (értelmező szótár) célszerű dokumen tálni.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 40 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 41 ►

A követelményeknek funkcionális, nem funkcionális és szakterületi szempontok alapján történő elkülönítése abban játszik nagy szerepet, hogy átgondoljuk

• a tervezett rendszer működését, a működési körülményeket (előírások, szakterületi szabályok stb.)

• milyen funkcionalitást, milyen felületet biztosítson a leendő szoftver a felhasználó felé, milyen belső funkciókat kell teljesítenie ahhoz, hogy a felhasználó igényeit kielégítse, és működés közben milyen előírásokat, szabályokat kell alkalmaznia, betartania.

**Felhasználói célok és követelmények leírása**

A SWEProducts olyan szoftverrendszer kifejlesztésére adott megbízást, amelyik biztosítja az árajánlatok készítését, a megrendelésekkel (az értéke sítéssel) kapcsolatos feladatok ellátását.

Az alábbiakban egy olyan dokumentum részletet mutatunk be, amely összefoglalja, hogy a megrendelő milyen instrukciókat adott az árajánlat kezelési és a megrendelés (értékesítés) szolgáltatás megvalósítására vonat kozóan. A megrendelő által felsorakoztatott igények adják a felhasználói követelmények halmazát.

Az árajánlat kezelési szolgáltatásra vonatkozó felhasználói követelmé nyek:

• Az ügyfelek kérhetnek árajánlatot a cég ügyfélszolgálatánál, amiket a cég munkatársai állítanak össze, de igénybe vehetik az Internetes ár ajánlat szolgáltatást is.

• A cég megkülönbözteti a vásárlókat. Vannak az „átlagos” vásárlók, és a nagyvásárlók. A cég a nagyvásárlókat kiemelt ügyfélként tartja nyilván: – A nagyvásárlóknak a megrendelések pénzügyi teljesítésére 30 na pos átfutási idő áll rendelkezésre. Az „átlagosként” nyilvántartott vásárlók a helyszínen fizetnek, készpénzben vagy bankkártyás fize téssel teljesítenek.

– A nagyvásárlók a vásárlási volumen függvényében kedvezményt (törzsvásárlói kedvezmény) kapnak. A kedvezmény mértékének meghatározása és jóváhagyása a Kereskedelmi menedzser feladat körébe tartozik. Abban az esetben viszont, ha az ügyfél három ko rábbi megrendelését pénzügyileg nem teljesítette, a törzsvásárlói kedvezményét elveszíti.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 41 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 42 ►

• Az árajánlatok, az abban feltüntetett árak, kedvezmények az árajánlat készítésének idejétől számított hat hónapig érvényesek.

• Az ügyfeleknek lehetőségük van az interneten készített árajánlatok kinyomtatására.

A megrendelés (értékesítés) szolgáltatásra vonatkozó felhasználói köve telmények:

• A fejlesztendő szoftver képes az ügyfélrendelések (vásárlások) kezelé sére.

• Vásárolni csak a cég ügyfélszolgálatánál lehet. Interneten csak árajánlat készítésére van lehetőség.

• Amennyiben egy ügyfél rendelkezik érvényes árajánlattal, a rendszer ben kell lenni olyan funkciónak, amelyik az árajánlatot vásárlá si/megrendelési tranzakcióvá alakítja, konvertálja. Erre a megoldásra azért van szükség, hogy az árajánlatban szereplő adatokat a cég mun katársainak vásárlás esetén ne kelljen újból felvenni, rögzíteni.



A követelmények szerepe meghatározó a majdan működő rendszer minő sége szempontjából. Emiatt rendkívül fontos, hogy kiemelt figyelmet for dítsunk a szoftverfejlesztési folyamat korai fázisában a követelmények teljes körű feltárására. Ügyelni kell azonban arra, hogy a követelményeket a fejlesztés kezdetén definiált célok alapján csak a megfelelő mélységig részletezzük ki.

A gyakorlat azt mutatja, hogy a fejlesztés során a megrendelőnek, a fel használónak újabb elképzelései, igényei merülhetnek fel. A korábban spe cifikált követelmények így a fejlesztés folyamán változhatnak, módosul hatnak. A modern rendszerfejlesztési módszertanok szerint (így a RUP

szerint is) a rendszert fel kell készíteni a követelmények változásának kö vetésére.

A követelményspecifikáció során az igények változásából adódó mó dosításokat, ill. a felmerült új igényeket első lépésben elemezni kell, majd meg kell vizsgálni, hogy a felmerült változtatások és az új igények milyen hatással lesznek a már felállított követelményrendszerre. A vizsgálat ered ményének kiértékelése után lehet csak dönteni a változások megvalósítha tóságáról.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 42 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 43 ►

3.2. A use case modell kialakítása

A RUP módszertan szerint a követelményspecifikáció munkaszakasz egyik legfontosabb feladata a fejlesztendő rendszer működését leíró use case modell elkészítése. A use case modellben a rendszer működésére vonatko zó funkcionális követelményeket use case-ek formájában írjuk le, ill. use case-ekhez kapcsolva gyűjtjük össze. A use case modellezés alapjául szol gáló use case-eknek kulcsszerepe van a rendszerfejlesztési folyamat vala mennyi fázisában. Végigkísérik a teljes fejlesztési folyamatot. A use case modellt használjuk, finomítjuk tovább az elemzés, a tervezés, az imple mentáció és végül a tesztelés során. Továbbá a modellben felállított köve telményhalmaz az alapja a fejlesztés teljes életciklusában folytatott folya matos ellenőrzési és kiértékelési feladatoknak.

A követelményspecifikáció végére előálló use case modell a rendszer tervezett funkcionális működését, a rendszer viselkedését írja le a rend szert kívülről, a felhasználó szemszögéből nézve. A modell három építő elemet tartalmaz:

• *use case-ek* (szoftverfunkciók – szolgáltatások), amiket a fejlesztendő szoftverrendszernek meg kell valósítani,

• *aktorok,* akik/amik a rendszer határán kívül vannak, a rendszerrel kap csolatba kerülnek, hogy a rendszerrel feladatokat (szoftverfunkciók) hajtsanak végre,

• a *kapcsolat* az aktorok és use case-ek közötti viszonyrendszert definiálja.

A témával foglalkozó irodalom a use case modell kialakításakor első lé pésben a use case-ek feltárását, egy use case listát javasol elkészíteni. A gyakorlat, a szakmai tapasztalatok azonban azt mutatják, hogy sok esetben elég nehéz a use case-ek listájának meghatározása. Első lépésben köny nyebb megtalálni, azono-sítani a rendszerrel kapcsolatba kerülő szereplő ket, aktorokat. Ha már ismerjük az aktorokat, azokhoz egyszerűbb hozzá társítani az általuk kezdeményezett, ill. végrehajtott use case-eket (funkció kat). A tankönyv a gyakorlatot követve elsőként az aktorokra vonatkozó ismereteket tekinti át, majd a use case-eket részletezi.

3.2.1. Aktorok

A felhasználóval folytatott beszélgetések, a felhasználói célokat összefog laló dokumentumok alapján körvonalazódik, hogy mik, vagy kik az érde-

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 43 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 44 ►

keltek4 a rendszer határán kívül, amik/akik közvetlenül kapcsolatba kerül nek, kommunikálnak a leendő szoftverrendszerrel. Az érdekelteket a use case modellben aktornak vagy szereplőnek nevezzük. A szereplő elneve zés helyett gyakran találkozhatunk a szerepkör kifejezéssel is. A rendszer felhasználói ugyanis meghatározott feladatkört (szerepet, jogosultságot) betöltve lépnek kapcsolatba a rendszerrel, egy konkrét szerepkört betöltve

használhatják a szoftverrendszert és annak szolgáltatásait. Az *aktor* egy szerep, amit az érdekelt játszik/végrehajt a rendszerrel folytatott interakcióban. A rendszer szereplője, aktora valaki vagy valami a rendszer határán kívül, aki/ami kapcsolatba kerül a rendszerrel, azzal fel adatot hajt végre, vagy a rendszernek egy adott szolgáltatását veszi igény be. Az aktorok nem kizárólag személyek, lehetnek tárgyak, gépek berendezések, üzleti egységek, vagy a rendszerrel kapcsolatot létesítő va lamely külső rendszerek, rendszerkomponensek.

**Az aktorok sajátosságai**

• Egy felhasználó többfajta szerepet is játszhat/végezhet, többféle sze repkörben lehet; egy szerepkört több felhasználó is betölthet (a rend szerben sohasem lehet két azonos szerepkört betöltő aktor).

• Az aktoroknak a rendszerrel kapcsolatban igényeik vannak, feladatok végrehajtását kezdeményezik, vagy a rendszer által nyújtott funkciók, szolgáltatások megvalósításában vesznek részt.

A feladatok végrehajtását kezdeményező szereplőket *kezdeményező szereplő* nek, a funkció (use case) megvalósításában részt vevőket *résztvevő szereplő* nek hívjuk. Egy use case-t mindig csak egy aktor kezdeményezhet, egy use case megvalósításában viszont több aktor is részt vehet.

• Az UML modellben az aktor nem objektum, hanem egy osztályszerű modellelem, amit az UML classifier minősítéssel azonosít, <<actor>> sztereotípiával5 ír le;

• Az aktor grafikus szimbóluma egy pálcikaember (lásd 3.3. ábra).

4 A követelményspecifikáció során azonosított érdekeltek köre (pl. a cég ügyfélszolgálati munkatársai, mint a szoftverrendszer használói) nem feltétlenül, sőt sok esetben abszolút nem egyezik meg az üzleti modellezés során felállított érdekeltek (pl. a cég vezetése, amely tendert írt ki a szoftverrendszer fejlesztésére) listájával.

5 Sztereotípia: A modellelemek magas szintű tipizálása. Lásd részletesen a 4.3.5. alpontot.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 44 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 45 ►

aktor

(szereplő)

**3.3. ábra.** Az aktor UML-szimbóluma

**Az aktorok megtalálásának módja**

A felhasználói célokat összefoglaló dokumentumokból kikeressük a főne veket. A kereséskor a releváns szereplők meghatározása érdekében célsze rű arra koncentrálni, hogy:

• Ki/mi használja a rendszert?

• Kik működtetik a rendszert, kik felelnek a rendszer karbantartási és adminisztrációs feladatainak végrehajtásáért?

• Kinek a hatáskörébe tartozik a biztonságkezelés, rendszervédelem?

• Létezik a rendszerben folyamatfigyelés, monitoring folyamat (moni toring process), amelyik hiba esetén újraindítja a rendszert?

• Kommunikál-e az új rendszer más rendszerekkel? Stb. **A rendszer szereplőinek specifikálásra vonatkozó előírások** • a rendszerrel közvetlenül kapcsolatba kerülő, a rendszert használó

érdekelteket (személyek, dolgok, más rendszerek, rendszerkomponen sek) a feladatkörre, szerepkörre utaló névvel kell ellátni, azonosítani, • az aktorok neve egy tetszőleges karaktersor,

• az aktor nevének megválasztásakor ügyeljünk arra, hogy az aktor neve azonosítsa a use case-t kezdeményező, vagy a use case megvalósításá ban részt vevő szereplőt,

• az aktor nevét a szimbólum alá írjuk,

• a specifikációban röviden meg kell adni, hogy az aktor mit vár el a tervezett szoftverrendszertől, mi a felelőssége.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 45 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 46 ►



**Az aktorok azonosítása**

A feladat megfogalmazásából azonnal látható, hogy a rendszert potenciáli san az ügyfelek és a cég munkatársai fogják használni. A rendszernek biz tosítani kell, hogy az ügyfelek interneten állíthassanak össze árajánlatokat. A munkatársak az árajánlat-készítés mellett az ügyfélnyilvántartással, a megrendelésekkel (értékesítéssel) kapcsolatos feladatokat látják el. A ki emelt nagyvásárlók részére a törzsvásárlói kedvezmények kezelése a cég menedzserének felelősségi körébe tartozik. A számlák egy külön Számlá zási modulban készülnek. Ez azt jelenti, hogy meg kell oldani az új rend szernek a Számlázási modullal való együttműködését, kommunikációját. A korábban meghatározott felhasználói követelmények, és a fenti logikai levezetés alapján a rendszerben négy aktort definiálhatunk: a Kereskedő, az Ügyfél, a Kereskedelmi menedzser és a Számlázó rendszer aktorokat.

Ügyfél Kereskedő Kereskedelmi menedzser

Számlázó rendszer



A rendszer aktorainak, és azok elvárásainak ismeretében már relatíve könnyebb meghatározni a use case-eket.

3.2.2. Use case-ek

A use case-ek a fejlesztendő szoftverrendszertől a felhasználó által elvárt, megkövetelt viselkedés(ek)t, a rendszer által nyújtott szolgáltatásokat írják le. A use case fogalmának pontos definiálására az irodalomban számos terminológia létezik:

• Feladatok, funkciók kisebb vagy nagyobb egységeinek specifikálására szolgáló grafikus ábrázolási technika – a use case-ek valójában a rendszer funkcionalitását, viselkedését fejezik ki a rendszert kívülről szemlélve.

• A rendszerrel kapcsolatban feltárt use case-ek megadják a fejlesztendő rendszer külső képét.

• A rendszer kívülről látható funkciói, ún. kapcsolódási pontok a szoft verrendszert használók és a szoftverrendszer között.

• Tevékenységek specifikációja.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 46 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 47 ►

A *use case* a felhasználó és a számítógépes rendszer közötti interakció defi niálására szolgáló modellelem. Tipikusan a szoftver és a felhasználó (ak tor) között lezajló kommunikáció, üzenetváltás lépéseit írja le a felhasználó szemszögéből. Egy use case pontosan azt határozza meg, hogy a felhasz náló (aktor) MIT akar a szoftverrel végrehajtani, milyen célt kíván megva

lósítani, ugyanakkor nem tér ki a megvalósítás, a HOGYAN részleteire. A use case-ek definiálásához hozzátartozik a use case nevének meg adása, valamint egy leírás elkészítése, amely tartalmazza a use case-ben definiált működés végrehajtása során elvégzett lépéseket, megvalósított műveleteket. Ezt a leírást általában forgatókönyvnek nevezzük. A rend szerfejlesztés során ezt a leírást bővítjük, strukturáljuk, ill. elkészítjük a use case-t strukturáltan leíró aktivitási diagramot.

A követelményspecifikáció munkaszakaszban definiált use case-eket a szakirodalom *fekete doboz use case*-eknek *(black-box use case)* is nevezi. A feke te doboz jelző azt hangsúlyozza, hogy a fejlesztésnek ebben a szakaszában nem térünk ki a rendszer, a rendszerkomponensek belső működésére. A cél csak a rendszer viselkedésének specifikálása a rendszert külső szemmel nézve. A követelményspecifikáció során folytatott use case modellezésnek fontos célja a rendszer határainak meghúzása, azaz annak definiálása, hogy a leendő rendszer milyen funkciókat tud végrehajtani, és melyeket nem. A 3.2. táblázat az ÁrajánlatotKészít\_Weben use case példáján keresztül de

monstrálja a rendszer szokásos külső viselkedésének leírását és a belső működés specifikálását.

**3.2. táblázat.** Példa egy use case esetén szokásos külső viselkedés leírásra és egy hozzá készített belső viselkedés leírásra

| **Black-box módszer a rendszer (külső) viselkedésének leírására** | **Belső működés specifikálása** |
| --- | --- |
| ÁrajánlatotKészít\_Weben use case: Végrehajtásakor az Ügyfél megadja az Árajánlat összeállításához szük séges adatokat, majd a rendszer elkészíti az árajánlatot. | ÁrajánlatotKészít\_Weben use case: A rendszer ellenőrzi a megadott adatokat, és ha az adatok helyesek, akkor a rendszer az adatbázisba, az árajánlat táblába beszúr egy új sort, rekordot. |

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 47 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 48 ►

**A use case-ekre vonatkozó jellemzők, sajátosságok**

• A fejlesztendő rendszer szempontjából megkülönböztetünk architek tirálisan fontos, egyéb és rendszeridegen use case-eket:

– Az *architekturálisan fontos use case-ek* az alkalmazás architektú-rájának meghatározása szempontjából kritikusak, szignifikánsak. Ezek a use case-ek valósítják meg a rendszer fő szolgáltatásait. Biztosítják a rendszer alapműködését, a felhasználó által definiált fő feladatokat.

– Az alap use case-ek mellett vannak ún. *egyéb use case-ek,* amelyek ke vésbé kritikusak a teljes rendszer működése szempontjából. A fej lesztés során a munkát célszerű a nagyobb prioritással rendelkező, architektúrálisan fontos use case-ek kidolgozásával kezdeni, majd később foglalkozni az ún. kevésbé kritikus, egyéb funkciókkal. Az architekturálisan fontos és egyéb use case-ek együttes halmaza adja meg a szoftverrendszer határát.

– A rendszer határának pontos meghúzása a követelményelemzés so rán történik meg. Ilyenkor előfordul, hogy a korábban definiált use case-ek közül néhány kikerül a rendszer végső követelményhalma zából. Ezekkel a use case-ekkel a fejlesztés további szakaszaiban nem szabad foglalkozni. A fejlesztendő rendszer szempontjából ezeket a use case-eket *rendszeridegen use case-ek*nek hívjuk.

• Egy use case lehet „kicsi vagy nagy”:

A use case-ek-hez készített forgatókönyvek (lásd később) a rendszer funkcionalitását különböző megközelítésben és különböző szinteken fejezik ki. Mivel egy use case absztrakciós szintjének mélységét mindig az határozza meg, hogy azt milyen céllal, és kinek készítjük, így bizo

nyos értelemben más-más részletezettséggel és pontosító információk kal készítünk use case-eket a felhasználókkal való kommunikációra, mást a fejlesztők közötti kommunikáció leírására. A RUP módszertan logikáját követve a use case-ek feltárása a részletes kidolgozási fázisban kezdődik, de azok részletes kifejtésére nem mindig ebben a fejlesztési szakaszban kerül sor. A use case-ek finomítására, a fejlesztés későbbi szakaszaiban történő teljes mélységű kifejtetésére a RUP által követett iteratív fejlesztési paradigma ad lehetőséget.

• Egy use case konkrét célt teljesít, valósít meg az aktor számára: Ebben az értelemben a use case-eket kétfajta megközelítésben értel mezzük. Beszélhetünk rendszer interakciókról (a rendszer interakciók megnevezése szinonim a korábban használt funkcionális követelmé-

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 48 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 49 ►

nyek elnevezéssel), és megkülönböztetünk felhasználói célokat. A use case-ek ilyen fajta megkülönböztetésére az ad lehetőséget, hogy a defi niált use case-ek milyen alapossággal írják le a követelményeket, a megvalósítandó célokat. A 3.3. táblázatban olvasható példarészlet egy szövegszerkesztő szoftver fejlesztésekor felmerülő potenciális use case-eket tartalmazza, elkülönítve a felhasználói célokat a rendszer in terakcióktól (funkcionális követelmények).

**3.3. táblázat.** Rendszer interakciók és felhasználói célok [12]

| **Felhasználói célok – user goals** | **Rendszer interakciók –**  **system interaction** |
| --- | --- |
| – „ensure consistent formatting for a document”;  – „make one document’s format the same as another”; | – „define a style”; „change a style”;  – „move a style from one  document to another”; |
| A fenti kettőség nem minden esetben határolható el egzakt módon: *„in dex the document”;* | |

• A use case-eket minden esetben az aktorok kezdeményezik: Az informatikai alkalmazásokat a felhasználók (a rendszer aktorai) működtetik, használják feladatok végrehajtásához. A fejlesztés során a felhasználók szoftverrel szembeni igényeit use case-ek formájában ír juk le. Az elérendő célok szempontjából releváns use case-ek megtalá lására a gyakorlatban számos módszer létezik. Megoldással szolgálnak: – az adott területre jellemző felhasználóval való folytatott közös be szélgetések, interjúk,

– kérdőívek használata csoportos felmérés esetén,

– brainstorming (ötletbörze) módszer alkalmazása. Használata első sorban új fejlesztések esetén hasznos, vagy nehezen megfogható, leírható problémák megoldásakor.

– vitakurzusok a korábbi beszélgetések során definiált dolgok (fel adatok, funkciók) megvitatására, tisztázására,

– egyszerű, ún. favágó módszer: a célokat megfogalmazó dokumen tumokból kigyűjtjük az igéket.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 49 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 50 ►

**A use case-ek specifikálásra vonatkozó szabályok**

• A követelményspecifikáció során meghatározott funkcionális követel ményeket leíró use case-eket a funkciójellegre utaló névvel kell ellátni, azonosítani.

• A use case-t azonosító név egy tetszőleges karaktersor. • A use case neve kettős szerepet tölt be. Azonosítja a diszkrét feladatot, amit a rendszernek teljesíteni kell, másrészt a megnevezés az adott use case-t meg is különbözteti a többi use case-től.

• A use case-eket (diszkrét feladat, funkció) az UML modellező nyelv szabályai szerint grafikusan egy ellipszis szimbólum jelöli (lásd 3.4. ábra). • A use case (funkció) nevét az ellipszis alá írva adjuk meg.

use case/funkció

**3.4. ábra.** A use case szimbóluma UML-ben

• Minden use case-hez tartozni kell egy use case leírásnak.

A fejlesztés során minden use case-hez készül egy részletes leírás. A rész letes leírásban a felhasználó szemszögéből rögzítjük a felhasználó és a rendszer között zajló üzenetváltás (párbeszéd) lépéseit. Egy use case azonban nemcsak a szokásos (normál) működéssel hajtódhat végre. A use case szokásos működését a use case normál lefutásának, más szóval alap folyamatnak hívjuk. A működésnek lehetnek különleges, alternatív esetei (pl. hibás működés), ezek az alfolyamatok. A fejlesztés során minden use case esetén fel kell tárni az összes alternatív lefutási menetet. A use case-ek

normál és alternatív működését külön forgatókönyvekben rögzítjük. A *forgatókönyv***6**, más szóval *szcenárió* a use case egy konkrét végrehajtása, lefutása, a use case egy példánya (instanciája). A use case-ben definiált működés lépésenkénti, ún. step-by-step végrehajtását tartalmazza a fel-

6 A forgatókönyv nem az UML és nem a RUP tartozéka. Már a hagyományos és korai OO-fejlesztésekben is alkalmaztak forgatókönyveket a feladatok végrehajtási lépéseinek leírásához, használatuk azonban informális jellegű volt. A forgatókönyvszerű leírások nagyban segítették a feladatok megértését, lehetőséget adtak a bonyolultabb problémák átgondolására, majd azok logikai strukturálására.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 50 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 51 ►

használó szemszögéből. Egy use case-hez az alternatív működések miatt több forgatókönyv készülhet, de egy biztosan. A forgatókönyv készítése kor a rendszer és a felhasználó között zajló üzenetváltásokat a felhasználó szerepében célszerű megfogalmazni, hiszen a felhasználó fogja az alkalma zást használni. Az üzenetváltások leírásakor a MIT-re koncentráljunk, azt írjuk le, hogy a use case működéskor MI történik, milyen tevékenységek zajlanak, és ne térjünk ki a HOGYAN részleteire, a megvalósítás módjára. A forgatókönyvben elsőként a use case-ben definiált működés normál végrehajtását írjuk le, de el kell készíteni az alternatív esetekhez tartozó forgatókönyveket is.

A forgatókönyv készítésére nincsenek szigorú formai előírások, meg kötések. A forgatókönyvben a use case működését, vagyis az egymás után zajló tevékenységeket szöveges formában, mondatszerűen fogalmazzuk meg. A forgatókönyv tartalma (egy lehetséges alternatíva):

• a feladat rövid értelmezése,

• az alternatív útvonalak meghatározása,

• a végrehajtásban résztvevő szereplők meghatározása, • a közös feladatok kiválasztása.

A forgatókönyv készítésekor érdemes megvizsgálni a következőket:

• Hogyan kezdődik a use case?

• Hogyan ér véget a use case?

• Milyen kölcsönhatások történnek az aktor és a use case között? • Milyen adatok cserélődnek a use case és az aktor között? • Milyen ismétlődő viselkedést hajt végre a use case?

A követelményspecifikáció munkaszakaszban a use case-ekhez készített forgatókönyvek összessége tisztán és érthetően modellezi a szoftver mű ködését. A use case-ek teljes részletezettségű kifejtésére az elemzés/terve zés fázisban kerül sor, ahol már a megvalósítás (use case realization) mi kéntjére fókuszálunk, a HOGYAN kérdésre helyezzük a hangsúlyt. Ebben a szakaszban térünk át az implementációs részletekre.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 51 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 52 ►



**Use case-ek definiálása**

A felhasználói követelmények és a rendszer szereplőinek ismeretében ha tározzuk meg a funkcionális követelményeket leíró use case-eket. Vegyük sorra a rendszerben definiált aktorokat, és határozzuk meg, hogy milyen use case-ek végrehajtását kezdeményezik, ill. mely funkciók végrehajtásá ban vesznek részt, működnek közre.

• A Kereskedő végzi az ügyfelek nyilvántartását, árajánlatot készít, szám lát állít ki, a megrendelésekkel (értékesítés) kapcsolatos feladatok fele lőse. A számlakészítés a Számlázó rendszerben történik. A számlaké szítéshez szükséges adatokat a Számlázó rendszer az új rendszerből

veszi, ezért meg kell oldani az új rendszer és a Számlázási modul kommunikációját is.

ÜgyfélTörzsadatotKezel

ÁrajánlatotKezel

MegrendeléstKezel Kereskedő

SzámlátKészítSzámlázó rendszer

• Az Ügyfél interneten árajánlatot tud összeállítani. Az Ügyfélnek, hogy igénybe vegye a cég webes árajánlat készítési szolgáltatását, regisztrálni kell magát a rendszerbe. A regisztráció során ki kell tölteni egy Re gisztrációs lapot. A regisztráció elfogadásával az Ügyfél bekerül a cég ügyfélnyilvántartásába. Az Ügyfél megtekintheti a korábban készített árajánlatokat (a rendszer az árajánlatokat csak a készítés időpontjától számított hat hónapig tárolja).

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 52 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 53 ►

Regisztrál

Ügyfél

ÁrajánlatotKészít

\_Weben

ÁrajánlatotLekérdez

\_Weben

ÁrajánlatotNyomtat

\_Weben

• A Kereskedelmi menedzser kezeli a cég kiemelt ügyfeleit, a nagyvásár lókat. A nagyvásárlóknak adott kedvezmények mértékének (meghatá rozása) jóváhagyása, törlése tartozik hatáskörébe.

KedvezménytJóváhagy

Kereskedelmi menedzser

KedvezménytTöröl

**Use case-ek finomítása**

A rendszer funkcionalitásának teljes mélységű feltárása, megértése érdeké ben a Kereskedő aktor által kezdeményezett use case-ek közül az Ügyfél TörzsadatotKezel use case-t tovább kell részletezni. A finomítás eredmé nyeként egy alsóbb szinten az ÜgyfélTörzsadatotFelvisz, ÜgyfélTörzsada totKarbantart use case-eket definiáltuk.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 53 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 54 ►

ÜgyfélTörzsadatotKezel

ÜgyfélTörzsadatot

Karbantart ÜgyfélTörzsadatotFelvisz

Az ÜgyfélTörzsadatotKezel use case esetén végrehajtott finomításhoz hasonlóan az ÁrajánlatotKezel use case-t is tovább kell részletezni. A le képezés eredményeként az ÁrajánlatotKészít, az ÁrajánlatotMódosít, az ÁrajánlatotNyomtat és az ÁrajánlatbólRendeléstVégez use case-ek specifi kálhatók.

ÁrajánlatotKezel

ÁrajánlatotKészít

ÁrajánlatotMódosít ÁrajánlatotNyomtat

ÁrajánlatbólRendelést

Végez

Az ÜgyfélTörzsadatotKezel és az ÁrajánlatotKezel use case-eket a finomí táskor csomag elemként specifikáltuk.

**Felhasználói követelmények megvalósítása use case-ekkel** A követelményspecifikáció első szakaszában meghatároztuk a felhasználói követelményeket. A use case modell finomítása során a felhasználói köve telményeket egy vagy több use case valósítja meg. Tekintsünk feladatunk ból egy konkrét példát!

Az árajánlat kezelési szolgáltatásra vonatkozó felhasználói követelmé nyeket az alábbi módon definiáltuk (lásd 41. oldal):

• Az ügyfelek kérhetnek árajánlatot a cég ügyfélszolgálatánál, de igénybe vehetik az internetes árajánlat szolgáltatást is.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 54 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 55 ►

Az árajánlat szolgáltatásra vonatkozó felhasználói követelményeket a mo dellben:

• az ÁrajánlatotKészít,

• az ÁrajánlatotMódosít,

• az ÁrajánlatotNyomtat,

• az ÁrajánlatbólRendeléstVégez,

• az ÁrajánlatotKészít\_Weben,

• az ÁrajánlatotLekérdez\_Weben és

• az ÁrajánlatotNyomtat\_Weben use case-ek valósítják meg, implemen tálják.

A felhasználói követelmények, és az azokat megvalósító use case-ek kö zötti kapcsolatot a függőséggel (dependency) modellezzük*7*.

**Követelmények megvalósítása Enterprise Architect-ben**

**ud KövetelményekMegvalósítása**

EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un ÁrajánlatotKészít ÁrajánlatotMódosítÁrajánlatotNyomtat

EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un

EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un ÁrajánlatbólRendeléstVég

EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un

ÁrajánlatKészítés

EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un

ÁrajánlatotKészít\_Weben

EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un ÁrajánlatotNyomtat\_Weben ÁrajánlatotLekérdez\_Weben

EA 5.1 Unregistered Trial Version EA 5.1 Unregistered Trial Version EA 5.1 Un

7 A felhasználói követelmények, és az azokat megvalósító funkcionális követelményeket leíró use case-ek közötti kapcsolatot leíró diagram az Enterprise Architect 5.0 (EA) CASE fejlesztőeszközzel készült. Az EA lehetőséget nyújt a követelmények modellezésé re. Az eszköz a követelményeket a diagramban is jól látható négyzetes szimbólummal jelöli.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 55 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 56 ►

**Részletes leírás készítése use case-ekhez**

A use case lista összeállítása után minden use case-hez készíteni kell egy részletes leírást. Az ÁrajánlatotKészít\_Weben use case-hez készített rész letes leírásban négy forgatókönyvet kell részletezni:

• A use case normál lefutása szokásos működés esetén: a use case sike resen véget ér, ha az Ügyfél elfogadja az Árajánlatra vonatkozó feltéte leket.

A szokásostól eltérő működéskor a lehetséges lefutások, alternatív útvonalak:

• Az Ügyfél a felhasználói név és a jelszó megadásakor a MÉGSEM gombra kattint.

• Az Ügyfél a felhasználói név és a jelszó megadásakor az OK gombot választja, azonban a rendszer a megadott adatok ellenőrzésekor hibát talál. A megadott adatok vagy azért hibásak, mert az ügyfél azokat rosszul adta meg, vagy azért, mert nem regisztrált felhasználó. Ilyen esetben a use case véget ér.

• Az Ügyfél a rendszer által küldött Árajánlat elfogadása üzenet megerő sítésekor a MÉGSEM gombot választja.

**Forgatókönyv a normál működés leírására**

• Az Ügyfél a cég honlapján aktiválja az „Árajánlat-készítés” funkciót. • A rendszer megjeleníti a Login dialógusablakot, ahol kéri az Ügyfél felhasználói nevét, és jelszavát.

• Az Ügyfél megadja a felhasználói nevét, és jelszavát.

• A rendszer ellenőrzi (validálja) a megadott adatokat. Hibás adatok ese tén újra kéri az adatokat.

• A rendszer megjeleníti az „Árajánlat-készítés” felületet. • Az Ügyfél megadja a kért adatokat.

• A rendszer validálja a megadott adatok helyességét, ellenőrzi az adatok konzisztenciáját. Ha az Ügyfélnek nem sikerült érvényesen kitölteni a lapot, a rendszer mindaddig visszatér a laphoz, amíg azt az Ügyfél he lyesen ki nem tölti.

• A rendszer megerősítést kér az Ügyféltől az Árajánlat elfogadására. • A rendszer elmenti az Árajánlat adatait, és nyugtázó üzenetet küld a képernyőn keresztül az Ügyfélnek az Árajánlat készítésének sikeressé géről.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 56 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 57 ►

A forgatókönyv készítésekor ellenőriztük, hogy:

• A use case akkor kezdődik, amikor a cég honlapján az Ügyfél aktiválja az „Árajánlat-készítés” funkciót.

• A use case sikeresen véget ér, ha az Ügyfél elfogadja az Árajánlatra vonatkozó feltételeket, a rendszer elmenti az adatbázisba az Árajánlat adatait.

• Az Ügyfél és a rendszer között kölcsönhatások történnek akkor, ami kor például az Ügyfél a felhasználó név és a jelszó megadása után megnyomja az OK gombot. Helyes felhasználó név és jelszó megadása esetén a rendszer megjeleníti az „Árajánlat-készítés” felületet.

• Adatok cserélődnek a szoftverrendszer és az Ügyfél között akkor, amikor például az Ügyfél megadja a felhasználói nevét és jelszavát, vagy akkor, amikor az Árajánlat készítésekor az Ügyfél megadja az adatokat.

• A szoftverrendszer mindaddig kéri a felhasználó nevet és jelszót, va lamint az Árajánlat készítési lapon szereplő adatokat, amíg azok kon zisztencia és validitási szempontból nem helyesek.



A forgatókönyv a use case egy konkrét végrehajtásának lépéseit mutatja be. A forgatókönyvben meghatározott lépéseket az UML-ben tevékenysé geknek, aktivitásoknak nevezzük. A forgatókönyv tehát a tevékenységek sorozatát írja le szöveges formában. A forgatókönyvben meghatározott tevékenységek menetének grafikus szemléltetésére az UML diagramok közül az *aktivitási (tevékenységi)* diagramot használjuk. Egy use case-hez any nyi aktivitási diagram8 készül, amennyi a forgatókönyvek száma. Gyakran azonban az alternatív végrehajtásokhoz tartozó forgatókönyveket – amennyiben nem bonyolítja az aktivitási diagramot – egy közös aktivitási diagrammal írunk le.

A forgatókönyveknek aktivitási diagramokkal való szemléltetése azért célszerű, mert grafikusan áttekinthetőbbé teszi, hogy mi történik, milyen tevékenységek zajlanak le a use case végrehajtásakor.

Az aktivitási diagram egy kezdő- és egy vagy több végpontot tartalmaz. A use case végrehajtási menetében az elágazási/döntési (őrfeltétel) pontok tól függően több végpont is lehetséges. A gyakorlatban a diagram készíté-

8 Az aktivitási diagramokat a 8. fejezet ismerteti.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 57 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 58 ►

sekor csak egy végpontot célszerű feltüntetni. A kezdő és végpontok mel lett a diagram felsorakoztatja az aktivitásokat (tevékenységeket), grafikusan kiemeli az elágazásokat jelölő döntési pontokat, a szinkronizációs vonallal elkülöníti a párhuzamosan, az azonos időben zajló tevékenységeket.



**Aktivitási diagram készítése use case-hez**

Az ÁrajánlatotKészít\_Weben use case-ben definiált működés normál vég rehajtását leíró forgatókönyv alapján készítsük el az UML aktivitási diag ramot!

Az Ügyfél a Kft. honlapján aktiválja

az "Árajánlat készítés" funkciót.

A rendszer megjeleníti a Login dialógusablakot,

ahol kéri az Ügyfél felhasználói nevét, és jelszavát.

Az Ügyfél megadja a

felhasználói nevét, és jelszavát.

A rendszer ellenőrzi, validálja

a megadott adatokat.

[ Hibás adatok! ]

[ Adatok rendben! ]

A rendszer megjeleníti az

"Árajánlat készítés" felületet.

Az Ügyfél megadja a

kért adatokat.

A rendszer validálja a megadott adatok

helyességét, ellenőrzi az adatok konzisztenciáját.

[ Hibás adatok! ]

[ Adatok rendben! ]

A rendszer megerősítést kér az

Ügyféltől az Árajánlat elfogadására.

[ Árajánlat Elutasítva! ]

[ Árajánlat rendben! ]

A rendszer elmenti az Árajánlat adatait.

A rendszer nyugtázó üzenetet küld az Ügyfélnek a képernyőn az Árajánlat készítésének sikerességéről.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 58 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 59 ►

**Aktivitási diagram úszópályákkal (swimlane-ekkel) kiegészítve**

**Szereplő:Ügyfél Rendszer**

Árajánlat készítés megkezdése

**Az ügyfél a Kft. honlapján aktiválja az**

**"Árajánlat készítése" műveletet.**

**A rendszer rmegjeleníti a "bejelentkezés"**

**párbeszéd-ablakot, amelyen bekéri a**

**felhasználó azonosító nevét és jelszavát.**

**Az ügyfél megadja az azonosító nevét és**

**jelszavát.**

**A rendszer ellenőrzi a felhasználói nevet és**

**jelszót.**

Ellenőrzés

[Hibás adatok!]

**A rendszer megjeleníti az "árajánlat készítés"**

**lapot**

**Az ügyfél megadja az árajánlat**

**elkészítéséhez szükséges adatokat.**

**A rendszer ellenőrzi a megadott adatok**

**helyességét, és megvizsgálja azok**

**konzisztenciáját.**

Ellenőrzés

[Hibás adatok!]

[elutasítva]

**A rendszer megerősítést kér az ügyféltől az**

**árajánlat elfogadására** Elfogadás

**A rendszer elmenti az árajánlat adatait A rendszer nyugtázó üzenetet küld az**

**ügyfélnek a képernyőre az árajánlat**

**készítés sikerességéről**

Árajánlat készítés befejezése



A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 59 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 60 ►

3.2.3. Kapcsolatok

A use case modell harmadik építőeleme a kapcsolat. *Kapcsolat* alatt klasszi kus értelemben az aktorok és a use case-ek közötti kapcsolatokat értjük. Az UML-ben a rendszer szereplői és a use case-ek között definiált kapcso latok modellezésére a use case diagram szolgál. A kapcsolatot a diagram ban az aktorokat és a use case-eket összekötő vonal szimbolizálja. A vonal lehet irányított.

**Aktorok és use case-ek között értelmezett kapcsolatok típusa** A korábbi alfejezetekben tárgyaltuk, hogy egy use case-t aktor kezdemé nyezhet, aktivizálhat. A rendszer szereplői és a use case-ek közötti kapcso latot egy nyíl jelöli, mint ahogy azt a 3.5. ábra use case diagramja illusztrál ja. Az irányítás nem a kommunikáció irányát mutatja, hanem azt, hogy melyik aktor kezdeményezi az adott use case végrehajtását.

Az aktor és a számítógépes rendszer között alapértelmezésben kom munikációs jellegű kapcsolat van, a kommunikatív jelleget a kapcsolatot szimbolizáló nyíl felett <<communicate>> sztereotípiával jelölhetjük. A gyakorlatban azonban a sztereotípiát nem szokás külön kiírni a kapcsolat típusának klasszikus volta miatt.

Kezdeményez use case ő

szereplő

**3.5. ábra.** Kapcsolat ábrázolása kezdeményezés esetén

Egy feladat (use case) végrehajtásában több aktor is közreműködhet (lásd 3.6. ábra). A use case megvalósításában részt vevő szereplőket és a use case-t egyszerű (irányítás nélküli) vonal köti össze.

A use case diagramban az aktorok és a use case-ek között definiált klasszikus társítási kapcsolatok mellett léteznek ún. speciális kapcsolatok. Értelmezünk két aktor közötti kapcsolatot. Definiálhatunk use case-ek közötti speciális viszonyokat is.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 60 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 61 ►

Kezdeményező szereplő

use case

Résztvevő szereplő1

Résztvevő szereplő2

**3.6. ábra.** Kapcsolat szemléltetése közreműködés esetén

**Speciális kapcsolat két aktor között**

Két aktor öröklődési viszonyban állhat egymással. Öröklődési kapcsolat akkor jöhet létre, ha egy use case végrehajtásakor több szereplő is betölti ugyanazt a szerepet. Az öröklődési viszonyban két aktortípust különbözte tünk meg, az egyik a leszármazott, a másik az ős szereplő. A leszármazott minden use case-zel kapcsolatban van, amivel az ős szereplő kezdeményez kapcsolatot. Az ős szereplőnél definiált minden use case végrehajtását kezdeményezheti, de annál akár többet is. Az aktorok között értelmezett öröklődési viszony UML-ben történő grafikus leírását a 3.7. ábra szemlél teti.

Ő use case/funkció s szereplő

Leszármazott

szereplő1Leszármazott

szereplő2

**3.7. ábra.** Öröklődési viszony aktorok között

**Speciális kapcsolatok use case-ek között**

A use case-ek között három speciális viszonyt értelmezünk. Megkülönböz tetünk tartalmazást, kiterjesztést és öröklődést.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 61 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 62 ►

**Tartalmazás – include viszony**

• A szereplő által kezdeményezett (alap vagy normál) use case-ek végre hajtásában vannak olyan részek, lépések, amelyek mindegyik use case végrehajtásakor bekövetkeznek és azonos módon játszódnak le. Ér demes az azonos viselkedéseket egy külön use case-be kiemelni. A ki emelt viselkedést tartalmazott vagy rész use case-nek hívjuk. A tartal mazott elnevezés utal arra, hogy a tartalmazott use case az alap use case-nek szerves része. A tartalmazott use case végrehajtása feltétel nélküli, vagyis az alap use case végrehajtáskor mindig bekövetkezik, le játszódik.

• Az UML-ben az alap use case-t és a tartalmazott use case-t szaggatott nyíl köti össze.

• A szaggatott nyíl az alap use case-től a tartalmazott felé mutat. • A kapcsolat tartalmazás jellegét a szaggatott nyílon elhelyezett, francia zárójelek közé írt *<<include>> sztereotípiával* jelöljük. A leendő rendszer fejlesztése során definiált use case-ek között értelmezett tartalmazás viszonyt a 3.8. ábra use case diagramja illusztrálja.

alap/normál use case1<<include>>

<<include>>

Aktor tartalmazott (included) use case

alap/normál use case2

**3.8. ábra.** Include kapcsolat use case-ek között

**Kiterjesztés – extend viszony**

• A modellben lehetnek use case-ek, amelyek végrehajtási menetében bizonyos feltételek bekövetkezésekor a vezérlés egy másik use case nek adódik át. Ilyenkor a normál use case-nek egy bővített változata játszódik le. Mivel a normál use case viselkedésében a feltétel csak bi zonyos esetekben következik be, ezért a normál use case-t bővítő vi selkedést érdemes külön use case-ben leírni.

• A feltételt a követelmények specifikálásakor kell megadni.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 62 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 63 ►

• Az UML-ben az alap use case-t és a kiterjesztett use case-t szaggatott nyíl köti össze.

• A szaggatott nyíl a kiterjesztett use case-től az alap use case felé mutat. • Az extend viszonyt a szaggatott nyílon elhelyezett, francia zárójelek közé írt *<<extend>> sztereotípiával* adjuk meg. A normál use case vég rehajtása során a normál use case végrehajtási menetében definiált fel tétel teljesülésekor bekövetkező vezérlésátadás UML szerinti grafikus leírását a 3.9. ábra use case diagramja szemléleti.

<<extend>>

alap/normál use case Aktor kiterjesztett (extended) use case

**3.9. ábra.** Az extend kapcsolat

**Öröklődés – generalizáció**

• Az öröklődési viszony esetén a leszármazott use case örökli a normál use case viselkedését, kapcsolatait. A leszármazott az eredeti/normál use case viselkedéséhez hozzáadhat újabb viselkedéseket, sőt az eredeti use case viselkedését felülbírálhatja, felülírhatja.

• Az öröklődést az UML-ben egy telt fejű nyíl jelöl. A nyíl a leszárma zott use case-től az általánosított normál (ős) use case felé mutat. Use case-ek között értelmezett öröklődési viszony grafikus megjelenítését a 3.10. ábra use case diagramja illusztrálja.

use case Aktor leszármazott use case **3.10. ábra.** Öröklődési viszony use case-ek között

A speciális kapcsolatokkal a rendszerünkben zajló folyamatokat tudjuk még pontosabban modellezni. Az <<include>> minősítést azoknál a funkcióknál használjuk, amelyek végrehajtási menetében közös tevékeny ségek azonosíthatók. Az <<include>> sztereotípia megadásával a közös viselkedést a rendszerben csak egyszer kell modellezni. A gyakorlatban az <<include>> sztereotípiát általában hibaesemények kezelésének, a fel-

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 63 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 64 ►

használói visszalépéseknek, megszakításoknak és egyéb különleges részte vékenység sorozatoknak a leírására használjuk.

A use case-ek között értelmezett <<include>> és <<extend>> kap csolatok modellezésekor az általánosítás irányának jelölése ellentétes. A nyíl az általánosabb viselkedésű modellelemtől a specializáltabb felé mutat. Az ábrázolásnak ez a módja logikus, hiszen <<extend>> viszony esetén a bővített use case az általánosabb, <<include>> esetén pedig a normál use.

**Use case modell készítése, strukturálása**

A use case modell strukturálása során a modellben azonosított speciális kapcsolatok az alábbiak:

• A Kereskedelmi menedzser rendelkezik mindazokkal a jogokkal, ami vel a Kereskedő munkatárs. Ez a modellben azt jelenti, hogy a Keres kedelmi menedzser a Kereskedő összes kapcsolatát örökli. A Kereske dő által elindított összes use case-t kezdeményezheti, sőt további fel adatokat (KedvezménytJóváhagy, ill. KedvezménytTöröl use case-ek)

is végrehajthat.

Kereskedő Kereskedelmi

menedzser

• A megrendelések (MegrendeléstKezel use case) összeállításakor a rendszer nagyvásárlók esetén ellenőrzi, hogy az Ügyfél három korábbi megrendelését pénzügyileg teljesítette-e. Abban az esetben, ha pénz ügyi teljesítésekre vonatkozó feltétel nem teljesül, (Extention Point) a rendszer nem engedi rögzíteni az újabb megrendelést. A vezérlés a Kedvezmény zárolása use case-nek adódik át, aminek végrehajtása so rán figyelmeztetés megy a Kereskedelmi menedzser-nek, aki törli az Ügyfélnek adott törzsvásárlói kedvezményt (KedvezménytTöröl use case). A KedvezménytZárol use case normál esetben (ha a feltételek: nagyvásárló+pénzügyi teljesülnek) nem játszódik le, csak akkor, ha a feltételek (az Ügyfél nagyvásárló és az Ügyfél három korábbi megren delését pénzügyileg nem teljesítette) teljesülnek.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 64 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 65 ►

<<extend>>

KedvezménytZárol Kereskedő MegrendeléstKezel

• Interneten az ügyfelek árajánlatot csak akkor készíthetnek, vagy a meg levő ajánlataikat csak akkor kérdezhetik le, ha regisztrált ügyfelek a rendszerben. Ez azt jelenti, hogy minden alkalommal, új árajánlat felvi telekor, lekérdezésekor azonosítani kell magukat a regisztráció (Re gisztrál use case) során megadott helyes felhasználói névvel és jelszó val. Az ügyfél-azonosítás az ÁrajánlatotKészít\_Weben, az Árajánlatot Lekérdez\_Weben use case-ek végrehajtásakor azonos módon játszódik le, ezért ezt érdemes kiemelni egy külön use case-be (ÜgyfeletAzono sít\_felhasználó név&Jelszó use case).

ÁrajánlatotKészít \_Weben

<<include>>

Ügyfél

<<include>>

ÁrajánlatotLekérdez

\_Weben

ÜgyfeletAzonosít\_ felhasználóinév&Jelszó

• Az ÜgyfélTörzsadatotKarbantart use case örökli az ÜgyfélTörzsada totFelvisz use case viselkedését. Az ÜgyfélTörzsadatotFelvisz use case végrehajtása új ügyfelet rögzít a rendszerben. Az ÜgyfélTörzsadatot Karbantart use case végrehajtása az ügyféladatok rögzítése mellett még biztosítja az ügyféladatok lekérdezését és módosítását. Módosítás ese tén a korábban rögzített és módosított ügyféladatok felülíródnak.

Kereskedő ÜgyfélTörzsadatotFelvisz

<<generalization>>

ÜgyfélTörzsadatot

Karbantart



A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 65 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 66 ►

3.2.4. Use case diagram

A követelményspecifikációban feltárt use case-eket, szereplőket és a kö zöttük értelmezett kapcsolatokat *use case diagramban* ábrázoljuk. A diagram segíti a rendszer viselkedésének megértését és grafikus modellezését. A fejlesztendő rendszerrel kapcsolatban elkészített use case diagramok al kalmasak a tervezett rendszerrel szemben támasztott követelmények (use case-ek halmaza) meghatározására, leírására. A diagram szemléletes, köny nyen áttekinthető, logikája könnyen érthető, emiatt nemcsak a fejlesztők által végzett követelményelemzést könnyíti meg, de hatékony eszköz a felhasználókkal történő kommunikáció megkönnyítésére.



**Use case modell**

Az ügyféladatok, az árajánlatok és a megrendelések (vásárlások) kezelését modellező use case diagram részletet az alábbi ábra szemlélteti.

ÜgyfélTörzsadatot

Felvisz

<<generalization>>

ÜgyfélTörzsadatot

ÁrajánlatotKészít

ÁrajánlatotMódosít

ÁrajánlatotNyomtat

Árajánlatból

Karbantart

Kereskedő

MegrendeléstKezel

MegrendeléstVégez

Kereskedelmi menedzser

SzámlátKészít

<<extend>>

KedvezménytZárol

KedvezménytJóváhagy KedvezménytTöröl

Számlázó rendszer

Az Ügyfél által kezdeményezett use case-eket összefoglaló use case diag ramot a következő ábra foglalja össze.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 66 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 67 ►

Regisztrál

ÁrajánlatotKészít \_Weben

<<include>>

Ügyfél

<<include>>

ÁrajánlatotLekérdez

\_Weben

ÁrajánlatotNyomtat

\_Weben

ÜgyfeletAzonosít\_ felhasználóinév&Jelszó



3.3. A use case modell dokumentálása A use case modellezés során fontos a különböző modellelemek megfelelő dokumentálása. Minden elemhez készíteni kell szöveges specifikációt, leírást, szükség esetén a még jobb megértés érdekében részletesebb ma gyarázó kiegészítést kell adni.

A use case modell dokumentálásának menete:

• Aktorok azonosítása.

• Minden aktor esetén meg kell határozni, hogy mit vár el a rendszertől. • Use case-ek feltárása, use case lista összeállítása.

• Minden use case-hez részletes leírás készítése:

– Alternatív forgatókönyvek készítése a use case működésének leírá sára.

– Aktivitási/tevékenységi diagram készítése.

• Kapcsolatok meghatározása:

– Kapcsolatok aktor és use case között.

– Speciális kapcsolatok azonosítása.

• A rendszer funkcionalitásának, viselkedésének grafikus modellezése use case diagramok készítésével.

• A fejlesztés során menetközben módosult funkciók dokumentálása, az elfogadott módosítások átvezetése a követelménydokumentumba.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 67 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 68 ►

3.4. Use case realizáció – a use case-ek megvalósítása

A követelménymodellben a use case-ek viselkedésének vizsgálatakor csak a felhasználó és a szoftverrendszer közötti interakciók pontos leírására koncentráltunk. Az új rendszert kívülről vizsgáltuk, elemeztük, a MIT-re, a rendszer által megkövetelt szolgáltatások megfogalmazására helyeztük a hangsúlyt, nem tértünk ki a megvalósítás részleteire, a HOGYAN-ra.

A use case modellben definiált use case-eket az elemzési, majd a terve zési modellben tovább elemezzük, részletezzük. Ezekben a fejlesztési munkaszaka-szokban a rendszer belsejét térképezzük fel. Azt vizsgáljuk, hogyan lesznek a rendszertől elvárt funkciók, use case-ek megvalósítva, majd implementálva. Ennek a munkának egy fontos lépése, hogy a use case modellben definiált minden use case-hez el kell készíteni annak meg valósítását specifikáló use case realizációt (use case realization).

Minden use case-hez definiálhatunk az elemzési modellben egy use case realizációt. A *use case realizáció* a use case egy lefutásának részletes le írása a rendszeren belül. A use case realizáció a modellben egy olyan use case lesz, aminek a sztereotípiája <<use case realization>>. A modellben a use case realizációt szaggatott vonalú ellipszis szimbolizálja (lásd 3.11. ábra).

use case realization

**3.11. ábra.** A Use Case Realization UML szimbóluma

A követelménymodellben definiált use case és az elemzési modellben en nek megvalósítására szolgáló use case realizáció között függőségi viszonyt értelmezünk. Az asszociációnál erősebb megszorítású függőségi viszony nyal azt fejezzük ki, hogy a use case megvalósítása a követelménymodell ben meghatározott use case viselkedésétől függ. A követelménymodellben meghatározott use case viselkedésében bekövetkezett változás kihat a megvalósításra, módosítja az implementációt. A két elem között értelme zett függőségi kapcsolat felett a <<trace>> sztereotípia megadásával jelöl jük, hogy az irányítatlan végén szereplő modellelem a másik modellelem megvalósítása (lásd 3.12. ábra).

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 68 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 69 ►

<<trace>>

use case use case realization

**3.12. ábra.** Use case megvalósítása függőségi viszonnyal

A use case realizáció modellezésére az UML diagramok közül az ese ménykövetési (szekvencia) diagramot használjuk. A diagram tartalmazza a működéshez szükséges objektumokat és az objektumok közötti üzenetvál tásokat, azok időbeli sorrendjében. A use case-ben leírt működést a rend szerben definiált objektumok valósítják meg, mely objektumok egymással üzenetváltásokkal kommunikálnak. A use case funkciója az üzeneteken keresztül teljesül. Az üzenetek eljárás vagy függvény jellegűek lehetnek, ez utóbbiak visszatérési értékét is megadhatjuk. Az üzenetek paraméterez hetők.

Az elemzési szakaszban a use case egy részletes lefutását leíró ese ménykövetési diagramot a tervezési modellben tovább részletezzük. Ha bár a különböző munkaszakaszokban elkészített eseménykövetési diagra mok mindegyike az eredeti use case-ben leírt működést írja le, azonban a

tervezés során készített modellek tovább részletezik az elemzési modell ben leírt működést a kiválasztott implementációs környezetben megvaló sítható objektumok, ill. osztályok definiálásával és a köztük megvalósuló üzenetváltások leírásával. Az elemzési modellben nagyvonalakban definiált szoftverarchitektúra végleges formája a tervezési fázisban alakul ki. A ter vezési szakaszban pontosan definiálni kell a leendő rendszer felépítését, egyértelműen elhatárolva egymástól a rétegeket (üzleti logika réteg, alkal mazási réteg, adatbázis réteg stb.), és a rétegekbe tartozó objektumokat.

**Use case relaizáció készítése**

Az ÁrajánlatotKészít\_Weben use case lefutását a rendszeren belül az ÁrajánlatotKészít\_Weben use case realization valósítja meg.

<<trace>>

ÁrajánlatotKészít \_Weben

ÁrajánlatotKészít\_Weben realization

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 69 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 70 ►

**Eseménykövetési diagram készítése az elemzési munkaszakaszban** Az ÁrajánlatotKészít\_Weben use case realizációt az eseménykövetési di agrammal írjuk le. A diagram az UML 1.5-ös szabvány alapján készült. A diagramon ábrázoljuk a működésben megjelenő objektumokat és az ob jektumok közötti üzenetváltásokat, azok időbeli sorrendjében. Az elemzési szakaszban a use case realizációt leíró eseménykövetési diagram még nem olyan részletes, mint a tervezési modellben készített eseménykövetési diag ram. Az eseménykövetési diagram (szekvenciadiagram) leírása az 5.2. al pontban található.

: Ügyfél Főablak AlkalmazásVezérlő LoginDialog Ablak ÜgyfélTörzsadat "Az Árajánlat készítés" funkció választása

JogosultsagLekerdezes( )

megjelenít( )

"Felhasználói név" és "Jelszó" beírása

| OK gomb megnyomása |  |  |
| --- | --- | --- |
|  | ügyfelKereses( ) | megjelenit( ) |
| Adatok megadása |  |
|  | OK gomb megnyomása |  |
|  | Elfogadja az Árajánlatot? |  |
|  | Elfogad gomb megnyomása |  |
|  | Az Ön által készített Árajánlat létrehozva! |  |

Árajánlat Készítés Ablak

ÁrajánlatKezelő

Árajánlat

adatokEllenorzese( )

Létrehozás

create( )

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 70 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 71 ►

**Eseménykövetési diagram készítése a tervezési szakaszban** A diagram készítésekor az UML 2.0 szabvány leírását követtük.





A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 71 ►

Az UML Nyelv használata Use case modellezés A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 72 ►

A szoftverfejlesztési folyamatban a követelmények összegyűjtése, elemzé se és változásaik nyomon követése fontos feladat. Az ellenőrzési munka során azt kell megvizsgálni, hogy a követelmények, a követelményeket leíró modellek (use case modellek), dokumentációk (pl. fogalomszótár) stb. minden szempontból megfelelnek-e a felhasználó elvárásainak. Az ellenőrzési feladatok végrehajtását review-k (értékelő áttekintés) készítése vagy prototípusok készítése segíthetik.

A követelményspecifikációban az ellenőrzési munkának ki kell terjedni a teljes követelményhalmazra. A rendszert leíró követelményhalmaznak:

• Minden szempontból teljesnek kell lenni, azaz tartalmazni kell minden szolgáltatás pontos leírását, az ehhez szükséges információkat. • Konzisztensnek kell lenni, vagyis a rendszer leírásban, a szolgáltatások definícióiban, a felállított modellekben nem lehetnek ellentmondások. • Hibamentesnek kell lenni.

A rendszerre vonatkozó követelményhalmaz mellett ki kell térni a köve telmények egyedi vizsgálatára. Érdemes számba venni, hogy a vizsgálat tárgyát képező követelmény érthető-e, jól definiált, tesztelhető-e, ismert-e a forrása, megvalósítása esetleg sért-e valamilyen szakterületi követel ményt.

A követelmények ellenőrzése, az ellenőrzési munka eredményeként a változtatások végrehajtása, a javítások elvégzése elengedhetetlenül szüksé ges a további fejlesztési feladatok felhasználói igényeknek megfelelő vég rehajtásához.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 72 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 73 ►

4. Osztályok, objektumok,

osztálydiagram

Az osztálydiagram a legismertebb objektumorientált (OO) modellezési technika. Az OO-módszertanok legalapvetőbb eszköze, a rendszer objek tumait leíró osztályokat és a közöttük levő kapcsolatokat modellezi.

A fejlesztés különböző munkaszakaszaiban készülnek osztálydiagra mok. Ez nem újabb és újabb modellek megalkotását jelenti, a fejlesztés teljes ideje alatt egy alapmodellt vizsgálunk. Ezt az egy modellt fokozato san bővítjük a megfelelő részletekkel a fejlesztés menetében előrehaladva. A modell aktuális állapotát, érettségét az egyes szakaszokban készített osz tálydiagramokkal ábrázoljuk.

4.1. Osztálydiagramok használata a fejlesztés különböző szakaszaiban

Az osztálydiagramokat a fejlesztés különböző szakaszaiban más és más cél lal definiáljuk és használjuk, ennek megfelelően azok értelmezése is eltérő.

4.1.1. Osztálydiagramok az üzleti modellezésben Az *üzleti modellezés* során készített ún. *szakterületi osztálymodell* a szakterület valós elemeit ábrázolja, amelyek meghatározó szerepet játszanak a vizsgá lat tárgyát képező szervezet, szakterület működésében, a probléma megér tésében. A modell célja az elemek fogalmi szintre emelése, a közöttük értelmezett kapcsolatok és szerepeik meghatározása. A szakterületi osz tálymodell elemei az üzleti aktorok és üzleti entitás jellegű objektumok. A szakterületi osztálymodell elkészítésének alapja az *üzleti folyamatmodell (Bu siness Process Model),* a folyamatmodellezés során meghatározott üzleti use case-ek, és a use case-ek megvalósításában részt vevő üzleti szerep lők/aktorok (szerepkörök) és entitások. A szakterületi osztálymodellezés során azonosított üzleti entitásobjektumokat, az objektumok közötti kap csolatokat az *üzleti objektummodellben (Business Object Model)* foglaljuk össze.

Az üzleti modellek (szakterületi osztálymodell, üzleti objektum modell) és a követelményspecifikációban felállított use case modell az alapja, beme nete az elemzés/tervezés9 során fokozatosan bővített osztálymodelleknek.

9 A RUP az elemzést és a tervezést nem önálló munkafolyamatként definiálja, azokat a kétdimenziós modellben egy munkaszakaszként értelmezi.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 73 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 74 ►

4.1.2. Osztálydiagramok az elemzési munkaszakaszban Az elemzés kezdeti szakaszában a *szekvenciadiagramok* segítségével feltárjuk a use case-ek működéséhez (use case realization) szükséges objektumokat, meghatározzuk az objektumok között zajló üzenetváltásokat. Az objek tumok kapcsolati viszonyait, a kapcsolatokban játszott szerepeket az *együttműködési (collaboration) diagramban* ábrázoljuk. Az együttműködési diag ram, amelyet a gyakorlatban objektumdiagramnak is neveznek, csak a use case működésében azonosított objektumokat, a kapcsolatokat, szerepeket és az objektumok kapcsolódásának számosságát képes leírni. A szekven ciadiagramok és az együttműködési diagramok segítségével feltárt objek tumokat leíró osztályok, adataik (attribútumok), műveleteik és kapcsolataik közvetlen forrásként, bemenetként szolgálnak a fejlesztendő szoftveral kalmazás *elemzési osztály-modelljének* elkészítéséhez.

Az elemzési osztálymodell az elemzés során feltárt új elemek mellett tartalmazza az üzleti objektum modellben specifikált azon elemeket is, amelyek a tervezett szoftverrendszer működéséhez szükségesek10. Az üzle ti modellezés során felállított üzleti objektum modell nem más, mint az elemzési osztály-modell egy speciális, más megközelítésben értelmezett formája. Az üzleti objektummodell a szakterület objektumai alapján általá nosított kategóriákat, és azok kapcsolatait ábrázolja. Az itt azonosított objektumokat részletezzük, újabb jellemzőkkel bővítjük az elemzési osz tálymodellben a szekvencia és az együttműködési diagramok segítségével.

4.1.3. Osztálydiagramok a tervezési szakaszban Az elemzési osztálymodellt az elemzés/tervezés során továbbbővítjük, fejlesztjük. Részletezzük az osztályokat, pontosan deklaráljuk az attribú tumokat, műveleteket, értelmezzük a kapcsolatokat, azonosítjuk a speciális asszociációs viszonyokat. A fenti műveletek eredményeként áll elő a szoftveralkalmazás részletes osztálymodellje, amely a kódolás alapja. A szakirodalom ezt a modellt **tervezési osztálymodellként** azonosítja.

4.1.4. Az osztálydiagramok jelölésrendszere, perspektívák A fejlesztés különböző szakaszaiban (üzleti modellezés, elemzés, tervezés) készített osztálydiagramok jelölésrendszerükben számos hasonlóságot mutatnak. Az osztályok és az osztályok között értelmezett kapcsolatok ábrázolására az UML nyelv szimbólumait használjuk. Az osztálydiagramok

10 Az üzleti modellben definiált üzleti modellelemek nem mindegyike szerepel az elemzési modellben, csak azok, amelyek a működést meghatározzák.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 74 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 75 ►

céljukban, tartalmukban különböznek, az eltérő munkaszakaszokban felál lított osztálymodellekkel mást akarunk leírni, mást hangsúlyozunk. A fej lesztés kezdetén főként arra van szükség, hogy csak a probléma megérté séhez szükséges lényegi elemeket és azok kapcsolatait modellezzük, nem

törődve azok megvalósításával.

Az elemzési szakasz végén – amikor a probléma már világos, és min den követelményt részletesen elemeztünk és definiáltunk – hasznos készí teni egy összefoglaló osztálydiagramot, amelyik részletesebb. Ez a modell már szoftverkomponenseket, interfész specifikációkat tartalmaz, de nincs elkötelezve egyik programnyelvi környezetnek sem.

A tervezési fázis végén rendelkezésre kell állni annak az osztálymo dellnek, amelyik már pontosan egy bizonyos programfejlesztési környe zethez kötődik (például a kódolás a Java programfejlesztői környezetben történik).

A fenti gondolatmenetre alapozva számos módszertan az osztálydiag ramok három alapvető perspektíváját különbözteti meg, amelyekhez az osztálydiagramok modellelemeinek három különböző értelmezése kapcso lódik. Bár ez nem része az UML definíciójának, de hasznos abban az érte lemben, hogy mindig csak az aktuális probléma megoldására engedi a fej lesztésben résztvevő szakembereket koncentrálni.

• *Konceptuális (lényegi – conceptual) perspektíva:* Ezek az osztálymodellek segí tenek a probléma megértésben, a kommunikációban. A konceptuális modell csak a lényegi elemekre, az elemek közötti kapcsolatok leírására koncentrál. A diagramban ábrázolhatjuk a kapcsolat fokát, megadhat juk az asszociáció nevét, a különféle asszociációs viszonyokat, mint öröklődés, aggregáció, kompozíció. A lényegi elemek megtalálásnak egyik hatékony módszere, amikor a felhasználóval folyatott megbeszé léseket rögzítő dokumentumokban megkeressük a főneveket. A mo dell készítésekor nem vesszük figyelembe azt a szoftvert, amelyik majd ezeket az elemeket implementálja.

• *Specifikációs (specification) perspektíva:* Átmenet a konceptuális és az imp lementációs megközelítés között. Az osztálymodellek ezen típusa már a fejlesztendő szoftverre koncentrál. A use case-ek fizikai megvalósítá sának módját írja le, a megvalósítás vázát adja. A modellelemekhez fe lelősségeket határoz meg, de ezt nem kód szinten teszi. A specifikációs modell az alkalmazás struktúrájára, felépítésére összpontosít. A fejlesz tendő szoftver funkcionalitását tekintve bonyolult, elsőként célszerű

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 75 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 76 ►

azt logikus részekre bontani. A specifikációs aspektusban készített modell a funkcionális felosztás eredményeként keletkezett szoftver komponenseket, azok közötti kapcsolatokat (interfész), a kapcsolódás módját hangsúlyozza. Emiatt ebben a megközelítésben nem igazán osztályokról, hanem típusokról beszélünk. (egy Order-nek lesz felelős sége, hogy megmondja melyik Customer-ért felelős – az Order és a Customer közötti kapcsolatra koncentrál).

• *Implementációs (implementation) perspektíva:* Ezek az osztálymodellek már valódi, a későbbi rendszerben megvalósítandó osztályokat írnak le konkrét programnyelvi környezetben.

Az üzleti modellezők főként a valós elemeket leíró konceptuális modellt készítenek. Az elemzők, tervezők szoftverkomponensekre koncentrálnak, a fejlesztésnek ebben a szakaszában a kidolgozás részletezettségétől füg gően a specifikációs vagy az implementációs perspektíva érvényesül.

4.2. Objektum, osztály

Az *objektumok* a valós világ elemeit reprezentálják, modellezik. Például ezek a valós elemek a fizikai világban megjelenő konkrét egyedek, diszkrét entitások, amik lehetnek személyek, fizikai berendezések, elvont fogalmak (pl. kamatláb) stb. Az objektumok a valós elemekhez hasonló jellemzőkkel bírnak. Tulajdonságaik vannak, ismert az állapotuk, leírható a viselkedé

sük.

**Az objektum állapota**

Az objektumok életük folyamán a környezetükben levő más objektumok kal kapcsolatba kerülhetnek, egymásra hatással lehetnek. Az objektumok az őket ért hatásokra adott módon reagálnak, aminek eredményeként kü lönböző állapotokba kerülhetnek. A pillanatnyi állapotot az adott pillanat ban felvett tulajdonságértékek és az objektumnak a környezetében levő más objektumokkal megvalósított kapcsolatai határozzák meg.

**Mit takar az objektumok viselkedése?**

Az objektumokat különböző hatások érhetik, másrészt az objektumok maguk is hatással lehetnek más objektumokra. Az objektum viselkedése nem más, mint az általa végrehajtott tevékenységsorozat, reakció az általa előidézett vagy az őt ért hatásra. Az objektumok egymásra üzenetek for

májában fejtik ki hatásukat, viselkedésük egyik kifejezési módja a kommu nikáció. Az objektumok önálló, diszkrét entitások, ezért a környezetükkel

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 76 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 77 ►

folytatott kommunikáció csak bizonyos módszerek által alkotott felületen keresztül engedhető meg (egységbezárás11 fogalma), ezáltal a rendszer áttekinthetőbb és könnyebben módosítható lesz.

4.2.1. Az osztály

Az objektumok információt tárolnak (tulajdonságaikat leíró attribútumok formájában) és kérésre feladatokat ún. metódusokat hajtanak végre. Az objektum tehát adatok és metódusok összessége.

Az objektumokat a modellben az osztály modellelem írja le. Az *osztályok* az azonos tulajdonságokkal (attribútumok), viselkedéssel (metódusok) és kapcsolatokkal rendelkező objektumok csoportjának, halmazának leírására szolgálnak. Az objektum az osztály konkrét megjelenési formája, egyedi példánya (instancia). Az osztály tehát attribútumok és metódusok (függvé nyek, eljárások) gyűjteménye, a közös tulajdonságokkal és viselkedéssel rendelkező objektumokat írják le.

A fejlesztési munka végső célja a rendszert felépítő osztályok, és az osztályok közötti kapcsolatok feltárása, majd az osztályok forráskódhoz rendelése és kódolása.

Az osztályt az UML-ben egy három részre osztott téglalap modellezi (lásd 4.1. ábra):

• az osztály megnevezését a felső részben adjuk meg,

• a középső részben találhatók az attribútumok (tulajdonságok), • az alsó rész a műveleteket tartalmazza.

Osztálynév

attribútumNév

műveletNév()

**4.1. ábra.** UML osztályszimbólum

11 Egy tetszőleges objektum tulajdonságadatai csak az adott objektum metódusaiban láthatók, kívülről más osztály metódusaiból csak azok az elemek érhetők el, amelyeket az osztály engedélyez.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 77 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 78 ►

Az UML-ben a három részre osztott téglalap alapértelmezésben osztályt szimbolizál. Ha az osztály egy konkrét példányát, vagyis pontosan egy objektumot akarunk modellezni, annak jelölésére az osztályhoz hasonló téglalap szimbólumot kell használni12. A 4.2. ábra jól illusztrálja, hogy az objektumot az osztálytól az különbözteti meg, hogy az objektum neve alá van húzva. Lehetőség van csak az objektum nevét megadni az Objek

tumNév szintaxissal, vagy a téglalapban az ObjektumNév:OsztályNév szintaxissal jelölhetjük azt is, hogy az objektum pontosan melyik osztály hoz tartozik, melyik osztály példánya.

 **4.2. ábra.** Objektumok jelölése

**Az osztálynév megválasztása**

Az osztály elnevezésekor arra kell ügyelni, hogy a választott név tisztán, egyedien jellemezze az osztályt. A név adja az osztály identitását. A válasz tott név lehetőleg nagybetűvel kezdődjön, célszerűen egyes számú főnév legyen. Ha az osztálynév több szóból áll, célszerű a szavakat nagybetűvel kezdeni, vagy aláhúzást használni a szavak között.

4.2.2. Attribútumok

Az objektumok meghatározott tulajdonságokkal, jellemző sajátosságokkal ún. *attribútumokkal (attribute)* rendelkeznek. Az attribútum-érték az attribú tum egy konkrét előfordulása, egy adott pillanatban felvett állapot, az ob jektumpéldányban tárolt elemi adat. Az attribútum elnevezésekor a válasz tott név mindig utaljon az általa hordozott információtartalomra. Az attri

bútumokhoz az attribútum nevének megadásán túl további információk rendelhetők. Az attribútumok specifikációjának általános formája:

**[láthatóság] név [: típus] [= kezdeti érték] [{jelleg}]**

A láthatóság kifejezi, hogy az objektum különböző tulajdonságai, metódu sai mennyire fedhetők fel a külvilág számára. Az UML az osztályjellem zőkhöz (attribútum, művelet) három láthatósági szintet javasol:

12 Az UML-ben az aktivitási, a szekvencia és az együttműködési diagramok mindegyike tartalmaz objektumokat.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 78 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 79 ►

• a + public: a jellemző interfészen keresztül tetszőlegesen minden osz tály által elérhető, nincs szükség metódusra az eléréséhez, • a − private: csak az osztály látja, csak saját objektumon belül látszik. • a # protected: a jellemző csak saját objektumból és az utódokból lát szik, csak az adott osztályon érhető el, a gyermek (leszármazott) és a barát (friend13) osztályok látják.

A típus egyszerű adattípusokat jelent. A kezdeti érték az objektum készíté sekor beállítandó értéket jelenti. A jellemzők speciális tulajdonságokat mutatnak (pl. frozen).

Az attribútumok meghatározásakor figyelmesen kell eljárni, mert egyes adatok az elemezés/tervezés későbbi szakaszaiban maguk is objektumok nak minősülhetnek (pl. lakcím vagy dátum). Az ilyen attribútumok számá ra a modellben külön osztályt kell definiálni.

4.2.3. Műveletek

A *művelet (operation)* az osztály által nyújtott szolgáltatás. Feladat, tevékenység, amit az osztály végre tud hajtani. Az osztályok által nyújtott szolgáltatásokat elsősorban eseménykövetési diagramok üzenetei alapján azonosítjuk.

A művelet megvalósítása a metódus. Egy osztály minden konkrét ob jektuma azonos műveletekkel rendelkezik. A metódusok segítségével vég zünk műveleteket a tulajdonságokon, ill. módosíthatjuk azokat. A műveleteket:

• a művelet jellegének, nevének,

• a paraméterek nevének és típusának,

• a visszatérési értékeknek és típusuknak, valamint

• az alapértelmezett értékeknek a megadásával jellemezzük.

A műveleteknek két nagy csoportját különíthetjük el. A módosító művele tek azok a műveletek, amik megváltoztatják az adott objektum állapotát, ezért végrehajtásukra mindig figyelni kell. A lekérdező műveletek csak paraméterértéket kérnek más objektumoktól, nem változtatják meg a le kérdezés pillanatában megfigyelt állapotot. Az utóbbiak további jellemző je, hogy tetszőleges sorrendben hajthatók végre. A műveletek ezen két

13 Egy osztálynak barátja (friend) egy olyan metódus vagy akár teljes osztály, amely hoz záférhet az adott osztály minden – ill. deklarációtól függően néhány – mezőjéhez és metódusához.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 79 ►

Az UML Nyelv használata Osztályok, objektumok, osztálydiagram A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 80 ►

csoportját az UML szintaxis nem különbözteti meg, ezért ezt célszerű valamilyen módon a nevükben jelezni.

A műveletek specifikációjának általános formája:

**[láthatóság] név [(paraméter lista)] [: visszatérési érték típusa] [{jelleg}]**

4.2.4. Asszociáció

Az osztályok szolgáltatásaikat legtöbbször csak más osztályokkal együtt működve tudják biztosítani. Ez a megszorító jellegű állítás azt jelenti, hogy az osztályoknak egymással kapcsolatot kell létesíteni, egymással kapcsolat ban kell lenni.

Az *asszociáció (association)* az osztályok objektumai közötti kapcsolat le írása, absztrakciója. Az asszociációt, mint viszonyt megtestesítő fogalmat az osztályok közötti viszonyokra értjük. A kapcsolat fogalmat az objektu mok közötti viszonyra értelmezzük.

Az osztályok közötti asszociációs viszonyokat számos paraméterrel jelle mezhetjük:

• Két osztály vagy osztályok közötti kapcsolatot a *két osztályt összekötő vonal* reprezentálja.

• Az *asszociációhoz név rendelhető:* a nevet az osztályokat összekötő vonal fölé, középre helyezve írjuk.

• Megadhatjuk az osztályoknak az asszociációban játszott *szerepét:* min den kapcsolathoz két szerep rendelhető, amelyek az asszociáció két végén levő osztályoknak az adott asszociációban betöltött szerepére vonatkoznak. A szerepek definiálásával párhuzamosan általában meg adjuk a kapcsolat (asszociáció) irányát. A szerepeknek az osztályokhoz rendelése az attribútumokhoz hasonló funkciót töltenek be.

• A *kapcsolat fokának* megadásával jelölhetjük, hány objektum vehet részt az asszociációban *(multiplicitás)*14: a multiplicitás kifejezi, hogy az egyik osztály egy objektumához a másik osztályból hány objektum tartozik, vagyis kifejezi, hogy az osztályok objektumai milyen számosságban kapcsolódnak egymáshoz. A multiplicitás jelölése az UML-ben:

**1:** az adott osztály egy objektumához a másik osztályból pontosan egy objektum kapcsolódik

**0..1:** 0 vagy 1 objektum kapcsolódik, opcionális kapcsolat

14 A multiplicitáshoz kapcsoló másik fontos fogalom a kardinalitás, a számosság fogalma. A kardinalitás megadja az adott osztályban specifikálható előfordulások minimális, illetve maximális számát.

A dokumentum használata | Tartalomjegyzék | Felhasznált irodalom Vissza ◄ 80 ►