Rational Unified Process

Áttekintés

Vég Csaba

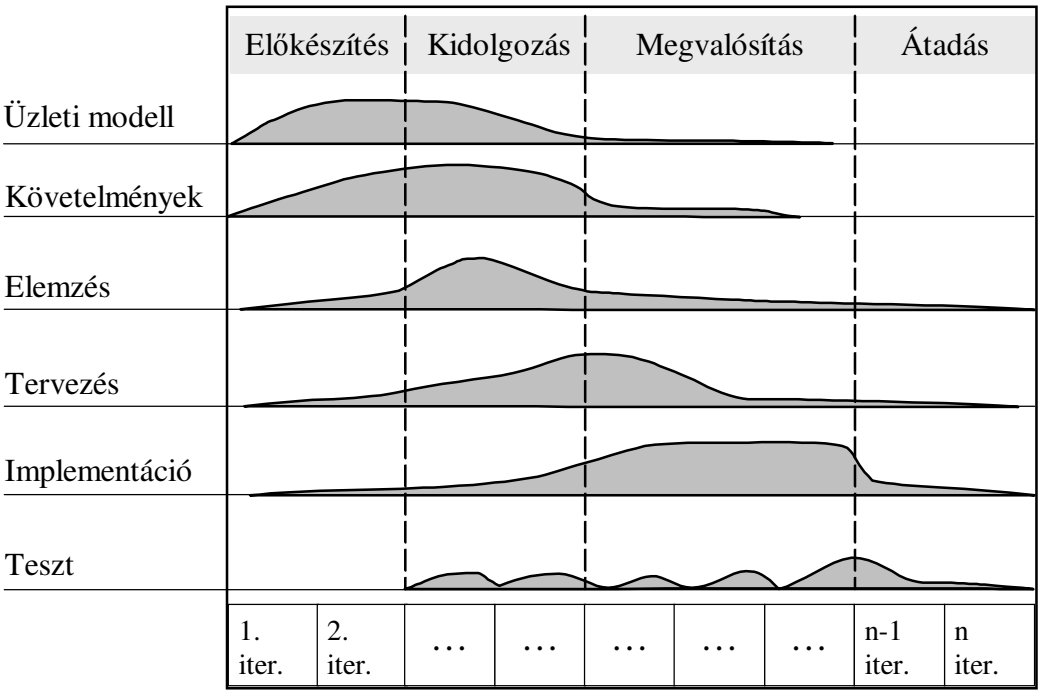
(www.logos2000.hu)

(2001)

A fejlesztés folyamata

A Unified Process a rendszerfejlesztés folyamatát alapvetően két dimenzióval írja le. Az egyik dimenzió a fejlesztés időbeliségét, dinamikáját követi. A másik dimenzió statikus szempontja az eljárás elemeit határozza meg, az elkészítendő dokumentumokat, diagramokat és forráskódokat, melyek közvetve az elkészítés lépéseit, munkafolyamatait is megadják.

Az időbeliség alapján a Unified Process a rendszerfejlesztést négy nagyobb egységre, négy *fázisra* bontja.



1. Ábra: Fázisok és munkafolyamatok

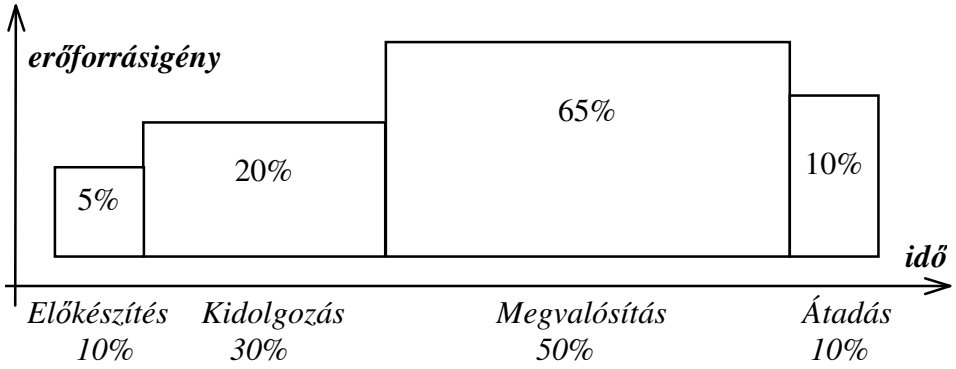
Az *Előkészítés* `(inception)` fázisában a rendszer eredeti ötletét olyan részletes elképzeléssé dolgozzuk át, mely alapján a fejlesztés tervezhető lesz, a költségei pedig megbecsülhetők. Ebben a fázisban megfogalmazzuk, hogy a felhasználók milyen módon fogják használni a rendszert és hogy annak milyen alapvető belső szerkezetet, architektúrát alakítunk ki.

A *Kidolgozás* `(elaboration)` fázisában a használati módokat, a „használati eseteket” részleteiben is kidolgozzuk, valamint össze kell állítanunk egy stabil alaparchitektúrát `(architecture baseline)`. A Unified Process készítőinek a képe alapján a teljes rendszer egy testnek tekinthető, csontváznak, bőrnek és izmoknak. Az alaparchitektúra ebből a bőrrel borított csontváz, mely mindössze a minimális összekötő izomzatot tartalmazza, annyit, amennyi a legalapvetőbb mozdulatokhoz elegendő. Az alaparchitektúra segítségével a teljes fejlesztés folyamata ütemezhető és a költségei is tisztázhatók.

A *Megvalósítás* `(construction)` során a teljes rendszert kifejlesztjük, beépítjük az összes „izomzatot”.

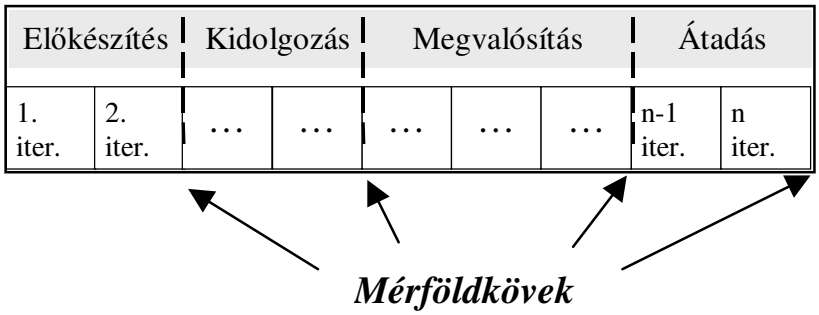
Az *Átadás* `(transition)` a rendszer bétaváltozatának kipróbálását jelenti, mely során néhány gyakorlott felhasználó teszteli a rendszert és jelentést készít annak helyességéről vagy a hibáiról és hiányosságairól. A rendszer javítása a rendszer módosítását, majd ezt követően újabb tesztelést jelent.

Minden fázis vége a fejlesztés egy-egy jól meghatározott *mérföldkövét* `(milestone)` jelenti, azaz olyan pontot, ahol egy célt kell elérnünk, illetve ahol kritikus döntéseket kell meghozni. Minden fázis végén megvizsgáljuk az eredményeket és döntünk a folytatásról.



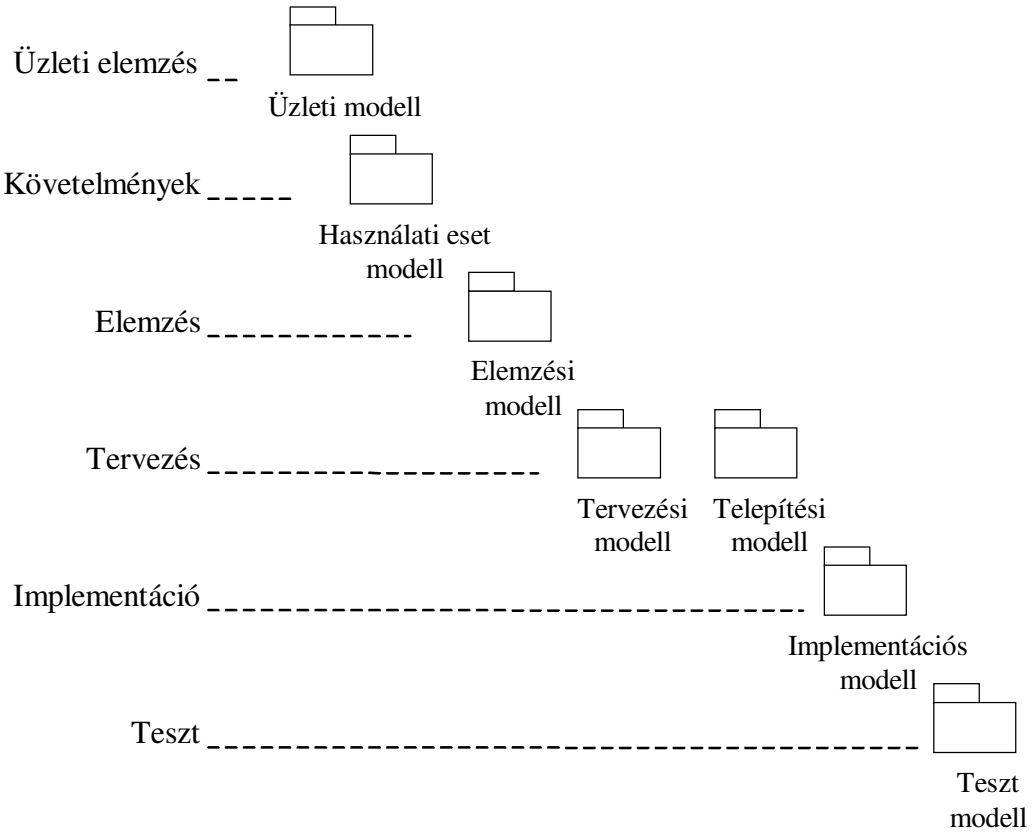
2. Ábra: Fázisok idő- és erőforrásigénye tipikus fejlesztés esetén

A fejlesztés nagyobb egységeit jelentő fázisok további kisebb egységekre, *iterációkra* `(iteration)` bonthatók. Minden iteráció egy teljes, illetve részben önálló fejlesztési ciklust jelent, mivel az iteráció végén egy működő és végrehajtható alkalmazásnak kell előállnia. Minden iteráció végén így a végső, teljes rendszer egyre bővülő részét kapjuk eredményül, melyeket a rendszer egymás utáni *kibocsátásainak* `(release)`, vagy belső változatainak nevezünk. A belső változatok lehetővé teszik, hogy azt a fejlesztők kipróbálhassák és annak tapasztalatai alapján esetleg módosíthassák a fejlesztés ütemezését.



3. Ábra: Mérföldkövek

A Unified Process felbontásában a fejlesztés menetének másik dimenziója az eljárás elemeit határozza meg, hogy a fejlesztés során milyen dokumentumokat, diagramokat, forráskódokat — összefoglaló néven *produktumokat* `(artifact)` — készítsünk el. Az elkészítendő produktumok természetesen a megfelelő tevékenységekkel állíthatók össze, mely tevékenységeket pedig adott szakismerettel rendelkező személyek („dolgozók”), adott sorrendben hajthatnak végre. A tevékenységek, azok időbeli sorrendje és az azt végrehajtó dolgozók együttesen egy munkafolyamattal `(workflow)` írhatók le.



4. Ábra: Modellek

Kezdetben a fejlesztéshez egy megfelelő kiindulópontot keresünk. Az első tevékenységcsoportunk így az *üzleti modellezés* `(business model)`, mely során megkeressük a készítendő rendszer üzleti vagy más néven szakterületi környezetét, mely alapvetően az üzleti fogalmakat és folyamatokat jelentik, illetve az azokra hatást gyakorló üzleti munkatársakat.

A következő tevékenység a *követelmények meghatározása* `(requirements capture)`. Ezen munkafolyamat során összegyűjtjük és felsoroljuk a rendszer működésével szemben támasztott kezdeti elképzeléseket, leírjuk azt, hogy a rendszernek milyen környezetben kell működnie, valamint felsoroljuk a funkcionális (működéssel kapcsolatos) és nem-funkcionális (pl. válaszidők, bővíthetőség, alkalmazott technológiák, stb.) követelményeket.

A követelmények meghatározása során alapvetően a felhasználók szempontjából írjuk le a rendszert, így annak egy külső képét rögzítjük. A következő munkafolyamat, az *elemzés* `(analysis)` folyamán a követelményeket a fejlesztők szempontjának megfelelően rendezzük át, így azok együttesen a rendszer egy belső képét határozzák meg, mely a további fejlesztés kiindulópontja lesz. Az elemzés során rendszerezzük és részletezzük az összegyűjtött használati eseteket, valamint azok alapján meghatározzuk a rendszer alapstruktúráját.

Az elemzés célja a szerkezeti váz kialakítása, mely vázat a következő munkafolyamat, a *tervezés* `(design)` formálja teljes alakká és tölti fel konkrét tartalommal, mely az összes — funkcionális és nem-funkcionális — követelménynek is eleget tesz. A tervezésnek az implementációval kapcsolatos összes kérdést meg kell válaszolnia, így

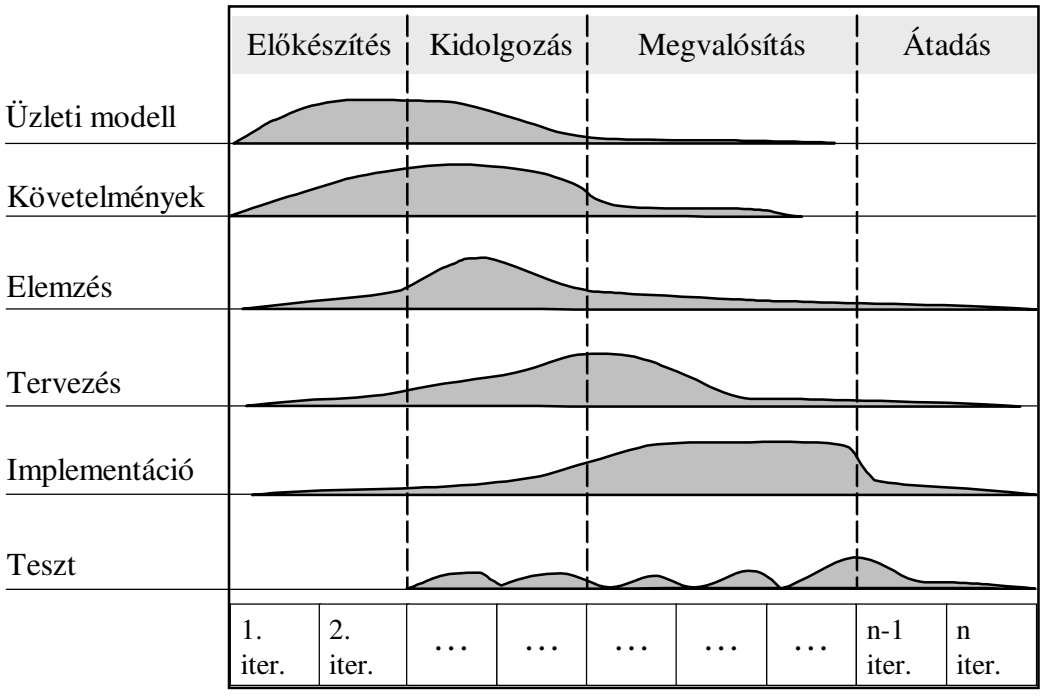
részletesen le kell írni az összes felhasznált technológiát, a rendszert független fejlesztői csoportok által kezelhető részekre kell bontani, meg kell határozni az alrendszereket és közöttük a kapcsolódási módokat, protokollokat. A tervezésnek a rendszert olyan részletezettségi szinten kell vázolnia, melyből az közvetlenül, egyetlen kérdés és probléma felvetése nélkül implementálható. A Unified Process szóhasználata szerint elő kell állítania az implementáció tervrajzát *(blueprint)*.

Az *implementáció (implementation)* során a rendszert az UML terminológiája szerinti komponensekként állítjuk elő, melyek forráskódokat, bináris és futtatható állományokat, szövegeket (pl. súgó), képeket, stb. jelentenek. Az állományok előállítása egyben azok függetlenül végrehajtható, önálló tesztjeit is jelentik. Az implementáció feladata még az architektúra, illetve a rendszer, mint egésszel kapcsolatos kérdések megválaszolása, így az iteráció esetén szükséges rendszerintegráció tervezése, az osztottság *(distribution)* tervezése.

Az utolsó munkafolyamat, a *teszt (test)* során összeállítjuk az iterációkon belüli integrációs tesztek és az iterációk végén végrehajtandó rendszertesztek ütemtervét. Megtervezzük és implementáljuk a teszteket, azaz teszt-esetekként megadjuk, hogy mit kell tesztelnünk, teszt-eljárásokként megadjuk azok végrehajtási módját, és programokat készítünk, ha lehetséges a tesztek automatizálása. A tesztek végrehajtásával párhuzamosan azok eredményeit szisztematikusan feldolgozzuk, majd hibák vagy hiányosságok esetén újabb tervezési vagy implementációs tevékenységeket hajtunk végre.

A felsorolt munkafolyamatok a Unified Process ún. *mérnöki elemei*, melyek mindegyike összefoglaló néven egy-egy *modellt* állít elő. A mérnöki elemek mellett léteznek még a fejlesztéssel kevésbé szoros kapcsolatban lévő *kiegészítő elemek* (pl. a fejlesztés irányítása) is.

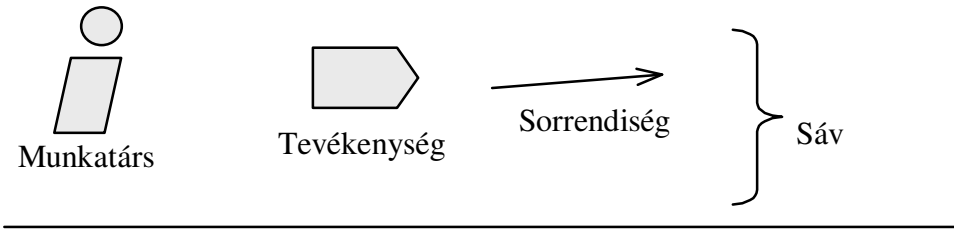
A fejlesztés folyamatának időbeni és munkafolyamatok alapján történő felbontása egy intenzitási görbe segítségével kapcsolható össze.



5. Ábra: Fázisok és munkafolyamatok

Az előkészítés fázisában a legnagyobb hangsúly a követelmények rögzítésére helyeződik, a többi tevékenység pedig jóval kisebb mértékben kap szerepet, tesztelés pedig gyakorlatilag nincs is. A későbbi iterációkban, illetve fázisokban ez a hangsúly fokozatosan áthelyeződik a technikaibb jellegű tevékenységekre. Az átadás fázisában már elsősorban tesztelés zajlik, így elemzés, tervezés és implementáció a tesztekre vonatkozóan és azok eredményeitől függően válik szükségessé, míg a követelmények összegyűjtése már nem kap szerepet.

A Unified Process az egyes munkafolyamatok *(workflow)* ábrázolására speciális diagramokat alkalmaz. A diagram alapelemei a tevékenységek, melyek mindegyike egy vagy több produktumot (dokumentumot, kódot, stb.) hoz létre, vagy azokon módosít. A diagramot a tevékenységeket végrehajtó személyek, ún. munkatársak alapján vonalakkal elválasztott sávokra *(swimlanes)* bontjuk. Minden tevékenység az azért felelős munkatárs mezőjében helyezkedik el, a tevékenységek ajánlott időbeli egymásutániságát, sorrendiségét pedig nyilakkal jelöljük.



6. Ábra: Munkafolyamat-diagram jelölései

Fontos megjegyeznünk, hogy a munkafolyamat-diagram a tevékenységeknek mindössze egy logikai sorrendiségét határozza meg, melytől a körülményeknek megfelelően eltérhetünk. Nem szükséges tehát követnünk a diagramot, hanem módosíthatunk a sorrendiségen, ha az ugyancsak az eredetivel „ekvivalens" végeredményhez vezet.

A munkatárs *(worker)* általában nem egy konkrét személyt jelöl, hanem egy adott szaktudást, illetve olyan pozíciót, amelyet egy vagy több konkrét személy is betölthet. A fejlesztés folyamán egyetlen személy különböző pozíciókban, különböző „munkatársként" is megjelenhet, illetve egyetlen „munkatárs" szerepét általában több konkrét személy tölti be.

A Unified Process jellemzői

A Unified Process más, objektumorientált rendszerfejlesztési eljárásokhoz a három legjellemzőbb tulajdonsága alapján hasonlítható.

A Unified Process az Objectory örököseként egy ún. *használati eset vezérelt (use-case-driven)* eljárás, amely azt jelenti, hogy a fejlesztés teljes folyamata során a használati eseteket eszközrendszerét, illetve az azzal kialakított modellt alkalmazzuk a fejlesztés ütemezésére.

A használati esetek önmagukban még nem adnak a teljes fejlesztési folyamathoz elegendő információt. A szükséges többletet a Unified Process a rendszer architektúrájaként nevezi meg, mely a rendszernek a fejlesztésben részt vevő összes munkatárs által közösen kialakított vázát, leglényegesebb elemeinek gyűjteményét jelenti.

A Unified Process *architektúra központú* `(architecture-centric)` eljárás, mivel a munkafolyamatokban nagy hangsúlyt kap a megfelelő architektúra kialakítása.

A Unified Process a fejlesztés teljes folyamatát kisebb részekre bontja, melyek önállóan is egy teljes fejlesztési folyamatot jelentenek. Az egyes részeket iterációknak nevezzük, mivel a fejlesztést ismétlődő, iteratív ciklusokban hajtjuk végre. Egy iteráció nem a rendszer egy, a korábbitól független részét állítja elő, hanem a rendszert újabb funkcionalitással bővíti, vagy a korábbi funkcionalitást teszi változatosabbá, gazdagítja. Az iteráció bővítését és módosítását inkrementumnak nevezzük. A Unified Process így egy *iteratív és inkrementális* `(iterative and incremental)` fejlesztési módszer.

Használati eset vezérelt eljárás

A Unified Process célja a fejlesztőknek egy olyan lépéssorozat megadása, mellyel hatékonyan kifejleszthetik és telepíthetik azt a rendszert, mely a felhasználók igényeinek megfelelő. A hatékonyság mérhető a költségek, a minőség és a fejlesztési idő egységeivel. A felhasználók igényeinek teljesítése és annak ellenőrzése azonban több problémába is ütközik. Részben magának a felhasználók igényeinek, a követelményeknek a felderítése is nehéz, az implementáció helyessége, a teljesítés ellenőrzése pedig a tesztelés feladataként jelenik meg.

A használati esetek eszközkészlete a gyakorlat által bizonyítottan az egyik legegyszerűbb és legalkalmasabb eszköz a rendszerrel támasztott alapvető igények összegyűjtésére. Mindezek mellett az így kialakult modell a teljes fejlesztési folyamat ütemezésére is kiválóan alkalmazható.

Az *elemzés* folyamatában a használati eset modellt vizsgáljuk, abból kiválasztjuk az aktuális iterációban kidolgozandó elemeket, majd abból egy olyan elvi („elemzési”) modellt építünk fel, amely logikailag alkalmas a használati esetek követelményeinek a megvalósítására. Ehhez elkészítjük a használati esetek elemzési szintű megvalósításait, melyek elvi szinten, osztályok közötti üzenetküldésekként „lejátsszák” a feladathoz szükséges interakciókat. A használati esetek, ill. a használati eset megvalósítások kettőse lehetővé teszi azt, hogy az elemzési modell minden egyes eleme esetén pontosan nyomon követhető, hogy milyen céllal került be a rendszerbe. A megvalósítások alapján a használati esetek feladatait az osztályokra vonatkozó követelményekké bontjuk szét.

A *tervezés* munkafolyamata során ugyancsak a használati eseteket vesszük sorra. Ekkor azok elemzési szintű megvalósításait tervezési szintű megvalósításokká írjuk át, melyeket már a kiválasztott konkrét platform és programozási nyelv eszközrendszerével és jelöléseivel adunk meg. A tervezés során így az elemzési elvi modellt konkrét gyakorlati tartalommal töltjük fel.

Az *implementáció* a tervezés leírásának megfelelő forráskódok és egyéb alkotóelemek, ún. komponensek elkészítését jelenti. A használati esetek ekkor segítenek kialakítani a komponensek implementációjának és integrációjának a sorrendjét.

Végül, a *teszt* munkafolyamat során a tesztelők ellenőrzik, hogy a használati esetek által meghatározott funkcionalitás valóban megfelelően és hiányok nélkül implementálásra került. A teszt modell a használati esetek alapján kialakított ún. teszt-eseteket tartalmaz, melyek adott bemeneti adatok és végrehajtási feltételek gyűjteményét, valamint az azokra adott elfogadható válaszokat határozzák meg.

Architektúra központú fejlesztés

A használati esetek rendkívül alkalmasak a követelmények összegyűjtésére és ábrázolására, valamint a fejlesztés ütemezésére, azonban önmagukban túlságosan „cseppfolyósak", nem rajzolják meg elég pontosan a készítendő rendszer körvonalait. A rendszer lényegi, működtető alapszerkezetét, a „körvonalakat" a Unified Process a rendszer architektúrájaként nevezi.

A rendszer architektúrája annak leglényegesebb elemeit tartalmazza, melyek együttesen a rendszer alapszerkezetét adják meg. Az alapszerkezet a rendszer megértése és kifejlesztése során alapvető fontosságú. A Unified Process szerzői egy házépítés példája alapján magyarázzák el az architektúrát. A házépítés előtt különböző tervrajzokat készítünk, például az építményt megadó rajzot, valamint több más, például a vízvezeték-, fűtés- és elektromos-rendszert ábrázoló rajzokat. Az egyes részek konkrét megvalósítása szempontjából a rajzok csak vázlatok, azonban meghatározzák a szükséges alapvető felépítést. A különböző munkatársak által igényelt különböző rajzok a teljes „rendszer" más és más nézeteit `(view)` határozzák meg, mely nézetek együttesen adják meg az architektúrát.

Az architekturális leírásra több szempontból is szükségünk van. A napjainkban fejlesztett szoftver-rendszerek általában nagyméretűek, bonyolult környezetben helyezkednek el és rendkívül összetett a belső felépítésük és működésük. Az architektúra a bonyolult teljes rendszer egy áttekintő képét fogalmazza meg, így segíti a rendszer megértését.

Az architektúra a fejlesztés felbontására és ütemezésére is alkalmazható. A rendszert általában alrendszerekre bontjuk, melyek egymáshoz adott interfészeken keresztül kapcsolódnak. A felbontást követően meghatározzuk az alrendszerek kifejlesztésének sorrendjét és a fejlesztésért felelős csoportokat.

Mivel az architektúra segíti a teljes rendszer áttekintését, ezért arra is kiválóan alkalmazható, hogy az elemek általánosításával több helyen is alkalmazható komponenseket határozzunk meg, melyeket ezután az ún. újrafelhasználással az ismételt fejlesztés költségének a töredékéért beépíthetünk.

Ugyancsak az áttekinthetőség miatt a jó architekturális leírás a rendszer későbbi módosításait is megkönnyíti, mivel könnyebben meghatározható, hogy mit és milyen módon kell módosítanunk és az milyen erőforrásokat igényel. A megfelelően kialakított architektúra esetén pedig egy új funkcionalitás bevezetése a rendszer kisebb mértékű változtatásával is végrehajtható.

Az architektúra kialakítását a rendszerfejlesztés környezete is nagymértékben befolyásolja, például a következő tényezők:

- alkalmazott rendszerszoftverek, például operációs rendszerek és adatbáziskezelők,
- alkalmazott middleware, pl. az osztottság keretrendszere vagy a felhasználói felületek keretrendszere,
- milyen, már létező külső rendszerekhez kell illeszkedni,
- milyen külső vagy belső szabványokat alkalmazunk,
- általános szerepű nem-funkcionális követelmények,

- osztottság architekturális követelményei, pl. kliens-szerver architektúra alkalmazása.

A Unified Process külön definiálja az *alaparchitekúra* `(architecture baseline)` fogalmát, mely a teljes rendszer egy önállóan is működő, erősen egyszerűsített változata. A későbbi iterációk ezen alaparchitektúra alapján, illetve az köré építkeznek

Iteratív és inkrementális eljárás

A feladatok összetettsége miatt minden szoftver-fejlesztési eljárás során célszerű egyértelműen megfogalmazni a fejlesztés *mérföldköveit* `(milestones)`, hogy világosan látható legyen a mérföldkövek közötti *fázis* `(phase)` során elérendő cél. A fázisok nagyobb egységeit ugyancsak célszerű további kisebb lépésekre bontani, mely lépések a Unified Process esetén az inkrementumokat megvalósító iterációs lépések lesznek.

Iteratív és inkrementális fejlesztés esetén a teljes fejlesztési folyamat nem vízesésszerűen, egyetlen nagy egységben történik. Ehelyett minden belső kis fejlesztési ciklus esetén először „szervezünk egy keveset, elemzünk, tervezünk, implementálunk egy keveset, majd integrálunk és tesztelünk egy keveset” — a Unified Process szerzőinek már szlogenné vált receptje szerint.

Az iterációs ciklusokban a rendszer egyre bővülő részeit fejlesztjük ki. A „bővítményeknek”, azaz az inkrementumoknak a rendszerhez való kapcsolása így nem a teljes fejlesztés végén, egyetlen nagy lépésben történik, hanem azt a fejlesztés során folyamatosan kell végrehajtani.

Az iteratív és inkrementális rendszerfejlesztési eljárás legfontosabb előnyeit a következőkben foglalhatjuk össze:

- a fejlesztés kritikus és lényeges kockázatai hamarabb azonosíthatók,
- a megfelelő architektúra könnyebben kialakítható,
- segíti a változó követelményeknek megfelelően, könnyebben módosítható keretrendszer kialakítását
- a munkatársak hatékonyabban vehetnek részt a fejlesztésben.

Fázisok

A fejlesztést az idő alapján a fázisok nagyobb egységeire bontjuk. Minden fázis a fejlesztés egy-egy jól meghatározott mérföldkövét jelenti, azaz olyan pontot, ahol egy célt elértünk, illetve ahol kritikus döntéseket kell meghozni. Minden fázis végén megvizsgáljuk az elért eredményeket és döntünk a fejlesztés folytatásról.

Minden fázis közbülső iterációkra bontható, mely egy-egy teljes fejlesztést jelent, amelyek mind végrehajtható alkalmazást, a végső, teljes rendszer egyre bővülő részeit eredményezik.

Előkészítés

Az előkészítés *(inception)* elsősorban üzleti szempontból írja le a fejlesztést és meghatározza az alkalmazás határait. Az üzleti szempont röviden a következőket jelenti:

- sikertényezők meghatározása
- kockázati tényezők felmérése
- erőforrás-becslés
- projekterv: a mérföldkövek dátumainak meghatározása

A fázis végén meghatározzuk az egyes iterációs ciklusok célját és döntünk a folytatásról

Kidolgozás

A kidolgozás *(elaboration)* során a problémát elsősorban szakterületi szempontból elemezzük. Ehhez a következő feladatokat kell végrehajtanunk:

- megbízható architektúrát alakítunk ki
- meghatározzuk a projekt-tervet
- megszüntetjük a legkritikusabb kockázati tényezőket

Az architektúrára vonatkozó helyes döntéseket csak a teljes rendszer megismerése után hozhatunk, ezért szükséges, hogy a használati esetek döntő részét specifikáljuk, valamint, hogy meghatározzuk a nem-funkcionális követelményeket.

Ezután a megoldási módokat ellenőrizzük egy olyan rendszer implementálásával, amely bemutatja a kiválasztott architektúra működését és végrehajtja a jellemző használati eseteket.

A fázis végén kiválasztottuk az architektúrát és megszüntetettük a főbb kockázati tényezőket. A mérföldkőnél itt is elemezzük az elért eredményeket és döntünk a folytatásról

Megvalósítás

A megvalósítás *(construction)* során a teljes rendszert iteratív és inkrementális módon kifejlesztjük. Ehhez:

- specifikáljuk az elmaradt használati eseteket
- a tervezésre helyezzük a hangsúlyt
- kiegészítjük az implementációt
- teszteljük az elkészített alkalmazást

A fázis végén már működőképes szoftvert állítunk elő, melyről döntenünk kell, hogy az kibocsátható-e a felhasználóknak.

Átadás

Az átadás *(transition)* lezáró fázisa során az alkalmazást átadjuk a felhasználóknak.

- Ez a fázis tipikusan az alkalmazás béta-tesztjével kezdődik.
- A rendszer hangolása miatt szükség lehet kiegészítő fejlesztésekre.
- A megjelenő hibákat ki kell küszöbölni.
- A még hiányzó részeket ki kell fejleszteni.

A fázis végén dönteni kell, hogy elértük-e a fejlesztés során kitűzött célokat, illetve, hogy indítanunk kell-e újabb fejlesztési ciklust. Ugyancsak célszerű, hogy a projekt tapasztalatait elemezve vizsgáljuk meg azt, hogy miként módosíthatunk a fejlesztési módszerünkön.

Összefoglalás

A Unified Process a rendszerfejlesztés folyamatát két dimenzióval írja le. Az egyik dimenzió a fejlesztés időbeliségét, dinamikáját követi, mely mentén négy fázist különböztet meg. Az Előkészítés fázisában a rendszer eredeti ötletét olyan részletes elképzeléssé dolgozzuk át, mely alapján a fejlesztés tervezhető lesz, a költségei pedig megbecsülhetők. A Kidolgozás fázisában a használati módokat, a „használati eseteket” részleteiben is kidolgozzuk, valamint össze kell állítanunk egy stabil alaparchitektúrát. A Megvalósítás során a teljes rendszert kifejlesztjük, az Átadás pedig elsősorban a rendszer bétaváltozatának a kipróbálását jelenti. A statikus dimenzió az eljárás elemeit határozza meg, az elkészítendő dokumentumokat (diagramokat, forráskódokat).

A Unified Process használati eset vezérelt, architektúra központú, valamint iteratív és inkrementális fejlesztési módszer.

Kérdések

- Mi a Unified Process megközelítésében a fejlesztés felbontásának két dimenziója.
- Az Előkészítés fázisában mi a fejlesztés célja?
- Melyek a Kidolgozás feladatai?
- Mi a teendő a Megvalósítás fázisában?
- Az Átadás fázisában mely tevékenységekre helyeződik a hangsúly?
- Milyen módon kapcsolódik össze a fejlesztés két dimenziója?
- Mit jelent az, hogy a Unified Process használati eset vezérelt?
- Mit jelent az, hogy a Unified Process architektúra központú fejlesztési módszer?
- Mit nevezünk iterációnak?
- Mi az inkrementum?

Üzleti modellezés

A rendszer tényleges fejlesztése előtt össze kell gyűjtenünk a rendszerrel kapcsolatos követelményeket. A követelmények pontos megértéséhez azonban az is szükséges, hogy ismerjük a készítendő rendszerrel kapcsolatos fogalmakat, azok viszonyait valamint az üzleti entitásokkal („objektumokkal") kapcsolatos műveleteket, azaz a rendszer környezetét, melyet üzleti környezetnek vagy üzleti modellnek is nevezünk.

Az *üzleti modellezés* `(business modeling)` lépése során a fejlesztés és egyben a kifejlesztendő rendszer környezetét határozzuk meg és írjuk le valamely formalizált módon. Az üzleti modellezés a későbbi munkafolyamatok egy előkészítő lépéseként, annak kiindulópontjaként szolgál.

Üzleti modell

Az üzleti modellezés eredménye az *üzleti modell*, mely alapvetően a szervezet (vagy szervezetek) dolgozóit, az azok által kezelt üzleti entitásokat és a kezelés módjait, azaz az üzleti használati eseteket tartalmazza.

Az üzleti modell egyrészről tartalmaz egy áttekintő szerepű dokumentumot, melyben összefoglaljuk az üzletmenet legfontosabb definícióit és céljait.

Az üzleti modell legalapvetőbb eleme a közös szótár vagy más néven *szójegyzék* `(glossary)`. A fejlesztés későbbi fázisaiban készülő dokumentumokban az egyértelműség és a konzisztens fogalmazás miatt csak a szótárban található elnevezéseket szabad használni. A szójegyzék rendkívül egyszerű felépítésű, fogalmanként mindössze annak megnevezését és rövid leírását tartalmazza.

A Unified Process az üzleti modellt alapvetően két megközelítésben tárgyalja. Az *üzleti használati eset modellben* a használati esetekként modellezett üzleti funkciókon van a hangsúly, mellyel a szervezeti működés alapvető szereplőit és azok felelősségeit ábrázoljuk. Az üzleti objektum modellben a főszereplők az üzleti munkatársak és az üzleti entitások, valamint a közöttük lévő kapcsolatok; a munkatársakat és entitásokat — csomagokként ábrázolt — szervezeti egységekbe, illetve szervezetekbe csoportosítjuk.

Az üzleti modellhez, azon belül is az üzleti objektum modellhez hasonló szerepű a *szakterületi modell* `(domain model)`, amely a környezet lényeges fogalmait szakterületi objektumokként jeleníti meg, és ez alapján ábrázolja a közöttük lévő kapcsolatokat, viszonyokat. Amíg az üzleti modell elsősorban az üzleti folyamatokra, tevékenységekre helyezi a hangsúlyt, addig a szakterületi modell célja az üzlet szereplőiről, objektumairól és fogalmairól egy szerkezeti-jellegű ábrázolás összeállítása. A szakterületi objektumok a későbbi elemzés és tervezés során általában a készítendő rendszerben típusokként, azaz osztályokként is megjelennek.

Munkatársak

Az üzleti modellezés munkafolyamatait végrehajtó munkatársakkal szemben fontos követelmény, hogy tisztában legyenek az üzlet szakterületének fogalmaival. További követelmény a jó kommunikációs képesség.

Az *üzleti folyamat elemző* `(business-process analyst)` vezeti és koordinálja a modellezendő szervezetre vonatkozó üzleti használati eset modell összeállítását, így annak megállapítását, hogy milyen üzleti aktorok és üzleti használati esetek léteznek és azok egymáshoz milyen módon kapcsolódnak.

Az *üzleti folyamat tervező* `(business designer)` feladata az adott szervezeti egység leírásának a részletezése, mely egy vagy több üzleti használati eset által alkotott munkafolyamat meghatározását jelenti. Megadja azokat az üzleti munkatársakat és üzleti entitásokat, amelyek az üzleti folyamatok („használati esetek") megvalósításához szükségesek. A tervező összegyűjti ezen munkatársak és entitások felelősségeit, műveleteit, attribútumait és a közöttük lévő viszonyokat.

Munkafolyamat

Szójegyzék összeállítása

A szójegyzék összeállítása során megkeressük és definiáljuk a szakterülettel kapcsolatos alapvető fogalmakat és kifejezéseket. Fontos, hogy az összes ezt követő dokumentumban konzisztens módon ezeket az elnevezéseket és azokat is csak a megadott definíció értelmében használjuk. Természetesen a fejlesztés során a szójegyzék bővülhet és módosulhat az újabb, illetve a tisztázódó követelményeknek megfelelően.

A következőkben példánkként szolgáló biztonság technikai szaküzlet információs rendszere esetén a szójegyzékünkbe tartozhatnak a következő fogalmak:

- **Szállítólevél:** Az üzletből a telepítők által elvitt árukról készített pozitív tételeket tartalmazó számla, amelyet tulajdonképpen a fizetési haladék igazolására használnak.
- **Visszavételezett szállítólevél:** Amikor a telepítő kifizeti a már elvitt termékek árát, akkor a bolt, az általa kiállított szállítólevelet visszaveszi.
- **Szállítólevél visszavételezéskor kiállított szállítólevél:** A szállítólevél visszavételezésének igazolására kiállítanak a szállítólevél tételeiről egy másik szállítólevelet, de ezt már negatív összeggel.
- **Beszerzési ár:** A szállítók által megállapított ár, amennyiért tőlük a szaküzlet beszerzi az árukat.
- **Végfelhasználói ár:** A szaküzlet által meghatározott ár.

A szójegyzékben a fogalmakat célszerű egyes számú főnevekként megadni. Az elnevezések definíciójában az azokkal kapcsolatba kerülő összes személynek egyet kell érteni, így a felhasználóknak és a fejlesztőknek is.

Üzleti aktorok és használati esetek keresése

Az *üzleti aktorok és használati esetek keresésének* `(find business actors and use cases)` lépésében az üzleti folyamatoknak a vázát állítjuk össze, így azt is meghatározzuk, hogy az üzlet mely részeit fogjuk modellezni, és mely részeket helyezzük

az érdeklődésünk határain túl. A vázlathoz összegyűjtjük azokat, akik kapcsolatban lesznek az üzlettel, majd a folyamatok alapján diagramokon ábrázoljuk az üzletmenetet.

Először *üzleti aktorokat keresünk*, mely jelenthet bármely személyt, csoportot, szervezetet, céget vagy akár gépet, mely kapcsolatban lehet az üzlettel. Néhány példa: vásárló, vevő, fogyasztó, megrendelő, partner, tulajdonos, befektető, külső információs rendszer… Amennyiben a modellezendő üzlet egy nagyobb vállalatnak a része, akkor ugyancsak aktorként jelenhet meg a vállalat többi egysége, illetve az ahhoz tartozó személyek által ellátott szerepek.

A példánk alapján ilyen aktorok a következők:

- Végfelhasználó: a szaküzlet árucikkeinek vásárlói, és a telepítendő rendszerek megrendelői.
- Telepítők: azok a vállalkozók, amelyek a bolt árukészletének felhasználásával biztonságtechnikai rendszereket építenek ki.
- Szállítók: azok a cégek, vállalkozások, gyártók, importőrök, amelyektől a szaküzlet az árucikkeket megrendeli, és beszerzi.
- Vállalkozásvezető: vele szemben elszámolás köteles az üzletvezető, és az ő hosszú távú irányelvei figyelembe vételével végzi az üzlet vezetését.

Az aktorok alapján ezután *üzleti használati eseteket keresünk*, mégpedig úgy, hogy megvizsgáljuk, hogy az egyes aktoroknak az üzletből milyen hasznuk származik. Célszerű az üzlet legfontosabb szereplőjével, a vevővel kezdeni, és megvizsgálni, hogy ő milyen alapvető szolgáltatásokat kap az üzletmenettől. Ehhez felhasználhatjuk a vevőnek az üzlet szempontjából vett életciklusát, azaz hogy milyen esetekben találkozik először az üzletmenettel, majd ezután milyen helyzetekben jelenhet meg. Ennél a lépésnél nagyon nagy segítséget nyújthat az üzletmenet szakértője azzal, hogy leírja az üzletmenet aktivitásait, majd ezeket üzleti használati esetekbe csoportosítja.

Példánk alapján a végfelhasználó használati esetei lehetnek a következők:

- Beszerzés igénylése
- Számlázás: számukra a bolt az értékesített termékekről számlát állít ki

A telepítőkkel kapcsolatos néhány használati eset a következő:

- Beszerzés igénylése
- Bizományi értékesítés: árucikkeket vihetnek ki az üzletből, azonnali készpénz fizetés nélkül, a bolt által kiállított szállítólevél kíséretében
- Számlázás: számukra a bolt az értékesített termékekről számlát állít ki, a visszavételezett szállítólevéllel egyidőben.

Az árajánlattétel használati esetét röviden a következőképpen írhatjuk le:

- Árajánlati adatok rögzítése, a szállító adatainak rögzítése és módosítása, valamint a termékek adatainak szükség szerinti rögzítése történik meg ennek az eljárásnak a végrehajtásakor. Az eljárást az új árajánlat érkezése, illetve a szállítóról érkező egyéb információ indítja.

Az üzleti aktorok, majd a használati esetek keresése után *meghatározzuk a használati esetek prioritásait*, ami elsősorban annak az eldöntését jelenti, hogy az lényeges szerepet kaphat-e az információs rendszerünk szempontjából, így azokat részletesen ki kell majd fejteni.

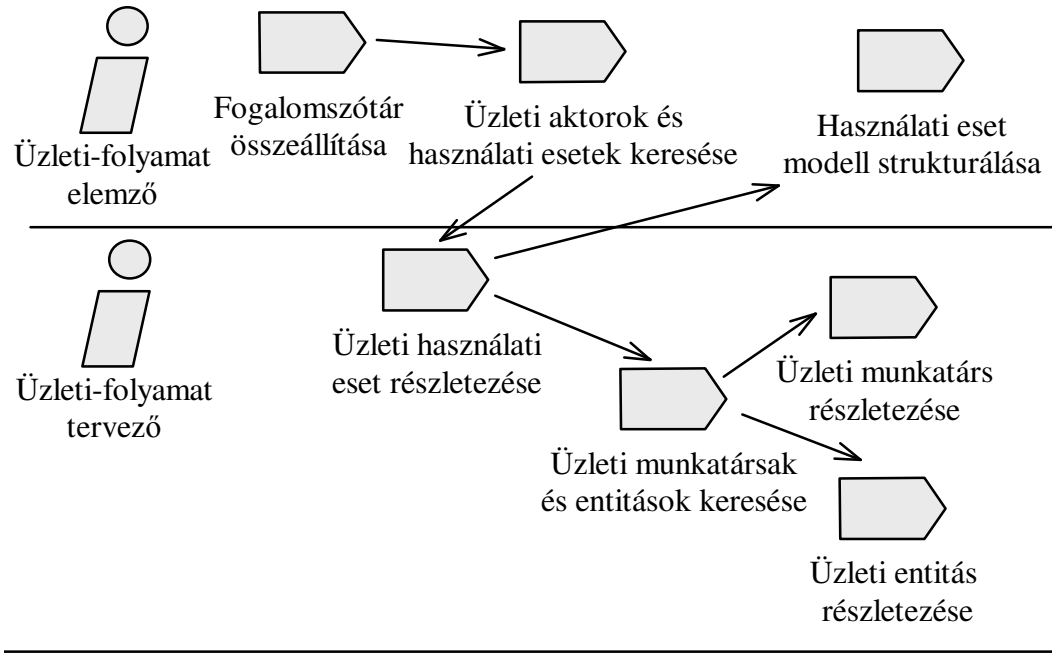
Az összegyűlt információk alapján *vázoljuk az üzleti használati esetek munkafolyamatát*. Ehhez az egyes használati esetekhez egy-egy tipikus lépéssorozatot írunk, mely lépései mind az aktor, illetve az üzletmenet egy-egy akciójára utalnak.

Az *üzleti aktorok és használati esetek interakciós módjának megadásakor* „kommunikációs" asszociációként jelöljük az elemek közötti kapcsolatot. Ha az aktor kezdeményezi a kommunikációt, akkor azt a navigációs irány megadásával hangsúlyozzuk.

Nagy mennyiségű üzleti használati eseteket és aktorokat *csomagokba csoportosíthatunk*, így a dokumentációnk érthetőbb és áttekinthetőbb lesz.

Ezután az *üzleti használati eseteket használati eset diagramokként ábrázoljuk*, mely az üzleti aktorokat, használati eseteket és a közöttük lévő viszonyokat szemlélteti.

Végül *elkészítjük az üzleti használati eset modell áttekintő összefoglalását*, melyben az üzletmenet legalapvetőbb céljait, illetve tipikus tevékenységsorozatokat gyűjtünk össze, valamint megemlítjük, hogy mely elemek *nem* fognak szerepelni a modellünkben.



7. Ábra: Üzleti modellezés

Üzleti használati eset részletezése

A kiindulási modellünkben a prioritások meghatározásakor már kijelöltük, hogy mely használati eseteknek szükséges az alaposabb vizsgálata, így az *üzleti használati esetek részletezésekor* `(detail a business use case)` ezeket vesszük sorra.

Kiindulási pontként *további információkat gyűjtünk a kiválasztott használati esetről*, mely alapja a használati eset munkafolyamatának lépésenkénti vázlata lesz, melyet tovább kell finomítanunk. A Unified Process ajánlása szerint meg kell neveznünk a használati esethez tartozó legalább tíz tevékenységet, illetve legalább tíz, az üzleti aktorral lefolytatott interakciót, mely utóbbi jelentheti az aktor kérését, vagy az üzletmenet reakciójaként megjelenő eseményt. Az összegyűjtött tevékenységeket és interakciókat az időbeliség mentén elhelyezve megkapjuk a munkafolyamat alapját.

Az üzleti használati eset részletezése során az alapként szolgáló munkafolyamatnak több változatát készítjük el. Először leírjuk a munkafolyamat egy vagy több normális lefolyását, majd ezután alternatív változatokat készítünk. Szöveges megadáskor fontos, hogy a szójegyzék fogalmait használjuk, egyben hivatkozhatunk is az ott szereplő leírásokra és fogalom-értelmezésekre.

Ha már a rendelkezésünkre áll az üzleti használati eset munkafolyamata, azt általában több részfolyamatra tudjuk bontani, azaz *strukturáljuk az üzleti használati eset munkafolyamatát*. Az üzleti használati eset folyamata több lehetséges útvonalat is követhet, adott aktor tevékenységétől vagy adott értékektől függően, illetve lehetnek benne opcionális, illetve párhuzamosan végrehajtható részek, melyek azonos időben is megtörténhetnek. Célszerű részfolyamatként kiemelni a tevékenységsorozat nagy, összefüggő részeit, így áttekinthetőbb lesz a leírás.

Az *üzleti aktorok és más használati esetekkel való viszonyt használati eset diagramon ábrázolhatjuk*, mely a kiválasztott használati esetre vonatkozó lokális diagram lesz.

A használati esetnél meg kell adnunk a *speciális követelményeket*, például a tevékenységek időkorlátait és a hatékonyság lényeges céljait, különösen azokat, amelyeket az információs rendszer szempontjából kell majd figyelembe vennünk.

Ha a használati esetet valamely más használati esetek kiterjeszthetik, akkor fel kell sorolnunk és meg kell neveznünk annak a *kiterjesztési pontjait*, azaz azokat a helyeket, ahol a kiterjesztések bekapcsolódhatnak a folyamatba.

Üzleti használati eset modell strukturálása

A folyamatosan bővülő használati eset modellünket ezen a ponton célszerű átdolgozni, hogy a további bővítésekkel együtt a részletezettebb változatnak is megtarthassuk az áttekinthetőségét. Az üzleti használati eset modell strukturálása *(structure the business use-case model)* jelenti bizonyos elemek kiemelését az alap használati esetekből, illetve új, általános aktorok felvételét.

Amennyiben valamely használati esetek munkafolyamatában találunk olyan nagyobb részeket, melyek kiemelhetők, akkor az így leválasztott részt az *include* kapcsolattal köthetjük az alap használati esethez. Ha a folyamat egy nagyobb része opcionális, azaz csak bizonyos körülmények között hajtódik végre, akkor azt kiemelve az alapesethez az *extend* kapcsolattal köthetjük. Ha bizonyos használati eseteknek a struktúrája, célja és működése hasonló, akkor azok alapján egy általános használati esetet vehetünk fel, melyet az általánosítás jelölésével kapcsolunk a speciális változatokhoz.

Gyakran előfordul, hogy adott üzleti aktorok bizonyos használati esetekhez mind formai, mind logikai szempontból azonos módon kapcsolódnak, azaz ugyanabban a szerepben jelennek meg. Ezt a helyzetet az aktor általánosításával szemléltethetjük.

Üzleti munkatársak és entitások keresése

Az üzleti használati esetek egyenként történő, részletező vizsgálatának folytatásaként ezután azokat egy másik szempontból kezdjük el vizsgálni. Ebben a lépésben üzleti munkatársakat és entitásokat keresünk *(find business workers and entities)*, így azonosítjuk az üzletmenettel kapcsolatos összes szerepet és „dolgot”, így le tudjuk írni, hogy ezek milyen módon vesznek részt az üzleti használati esetek megvalósításában.

Először azonosítjuk és leírjuk a *szervezeti egységeket*, kiválasztva azokat, melyek lényegesek lesznek a modellünk szempontjából. Amennyiben azok lényeges bemenetet

adnak, vagy lényeges kimenetet kapnak valamely használati esettől, akkor azok a rendszer környezetének aktoraként jelenik meg.

A szervezeti egységben minden egyes szerepet *üzleti munkatársként* kell azonosítanunk, melyeket mind egy-egy rövid leírással is el kell látnunk.

Az *üzleti entitások* azonosítását az üzleti munkatársak segítségével végezhetjük el, sorra véve, hogy a munkatárs milyen „dolgokat" kezel. Amennyiben az üzleti entitásoknak „tudniuk kell egymásról", akkor azt asszociációs kapcsolattal ábrázoljuk. Ha közöttük a viszony „rész-egész" jellegű, akkor azt aggregációs kapcsolatként jelöljük. Az általános entitás és annak egy vagy több speciális változata esetén az általánosítás-pontosítás viszonyt kell alkalmaznunk. A viszonyokat így végül egy osztálydiagramon ábrázoljuk.

Minden egyes üzleti használati eset sorravételénél meg kell határoznunk annak a *realizációját*, megvalósulását. Ehhez azonosítjuk a folyamatban részt vevő üzleti munkatársakat és entitásokat, majd a csoportmunkát egy együttműködési diagramon ábrázoljuk. Először célszerű a normál lefolyást ábrázolni, majd az alternatív és az opcionális változatokat.

Az elemek bővülésével az áttekinthetőség érdekében célszerű *strukturálni* az üzleti objektum modellt. Ehhez meghatározzuk minden egyes üzleti entitás életciklusát. Minden entitást egyszer elkészítenek, ill. végül törölnek az üzletmenet folyamán. Ellenőrizzük, hogy az entitást eléri, illetve használja-e valamely munkatárs. Általában az entitásnak létezik egy tulajdonosa, aki azért felelős — ez a viszony asszociációként modellezhető.

Üzleti munkatárs leírásának részletezése

Ebben a lépésben az összegyűjtött üzleti munkatársakról részletezettebb információkat adunk meg.

A részletezés legfontosabb eleme a munkatárs felelősség-területeinek a megadása. A felelősség-területeket ezután egy vagy több műveletté bontjuk szét, melyeket a munkatárs végre tud hajtani. A műveletek alapján ezután összegyűjtjük a munkatárs szükséges attribútumait.

Üzleti entitás leírásának részletezése

A következőkben az összegyűjtött üzleti entitásokat sorra véve, tovább részletezzük azok leírását.

A részletezést itt is a felelősség-területek leírásával kezdjük. Ehhez pontosan ismernünk kell az entitás életciklusát annak elkészítésétől a megszűntetéséig tartó változás lépéseit, mely folyamatot állapotdiagramként ábrázolhatunk. Az entitásra vonatkozó üzleti használati esetek megvalósításait vizsgálva össze tudjuk gyűjteni az entitás műveleteit, mellyel az aktorok arra hatást tudnak gyakorolni, illetve arról információt tudnak kérni. A műveletek alapján ezután összegyűjtjük az azok végrehajtásához szükséges attribútumokat.

Összefoglalás

A fejlesztés első nagyobb munkafolyamata az üzleti modellezés. A rendszer tényleges fejlesztése előtt ismernünk kell a készítendő rendszerrel kapcsolatos fogalmakat, azok viszonyait, valamint a műveleteket, azaz a rendszer környezetét, melyet üzleti környezetnek vagy üzleti modellnek is nevezünk. A fejlesztés során központi szerepű lesz az itt összeállított szójegyzék. Az üzleti modell egyrészt üzleti használati eset modellt

jelent, melyben az üzleti funkciókon van a hangsúly. A szakterületi modellben a környezet lényeges fogalmait és a közöttük lévő kapcsolatok ábrázoljuk.

Kérdések

- Miért lényeges szerepű az üzleti modellezés?
- Miért fontos a fejlesztés során a közös szójegyzék használata.
- Az üzleti használati eset modellben mire helyeződik a hangsúly?
- A szakterületi modell mit ábrázol?
- Melyek az üzleti modellezés munkatársaival szemben támasztott különleges követelmények?
- Milyen felépítésű a szójegyzék?
- Milyen lépésekben állítjuk össze a vázlatos üzleti használati eset modellt?
- Hogyan részletezhetjük az üzleti használati eseteket?
- Hogyan állítjuk össze az üzleti szakterületi modell vázlatát?
- Hogyan pontosíthatjuk az üzleti szakterületi modellt?

Követelmények

A rendszerfejlesztés során az egyik legnehezebb és egyben a legnehezebb körülmények között végrehajtott művelet a *követelmények tisztázása*, melyen annak a meghatározását értjük, hogy pontosan mit is kell kifejlesztenünk. A követelmények összegyűjtését általában az is nehezíti, hogy a rendszert a fejlesztők nem önmaguknak, hanem külső személyeknek, felhasználóknak készítik. Elvileg a felhasználók tudják, hogy milyen rendszert szeretnének, azonban tőlük nem követelhető meg, hogy az informatika fogalmainak megfelelően fogalmazzák meg az igényeiket, sőt általában az sem, hogy azokat pontos, formalizált módon és rendszerezve adják meg. Tovább bonyolítja a helyzetet, hogy a legtöbb rendszerrel több felhasználó is kapcsolatban van, akiknek a rendszer más és más vetületként, nem pedig teljességében jelenik meg.

A problémára a rendszerfejlesztések során az a megoldás született, hogy a felhasználók és a programozók közé elemzőket „iktatunk be", akik a felhasználókkal történő interjúk alapján segítenek a követelmények összegyűjtésében és megkísérlik abból előállítani az elkészítendő rendszer teljes, pontos és konzisztens képét. A tapasztalatok azt mutatják, hogy a felhasználók még az összeállított dokumentumok alapján sem képesek a készítendő rendszert megérteni és ellenőrizni, hanem csak akkor, amikor az már majdnem a befejezés előtt áll. A problémára részben megoldást nyújt az iteratív fejlesztés, mivel így a felhasználók hamarabb jutnak kipróbálható programrészekhez.

A követelmények tisztázásának az a célja, hogy meghatározzuk a készítendő rendszer határait. Össze kell foglalnunk azokat az igényeket, amelyeket teljesítenie kell a rendszernek és az egyértelműség miatt célszerű megfogalmaznunk olyan pontokat is, amelyek nem lesznek feladatai a rendszernek. A követelményeket — amennyire az csak lehetséges — egyeztetnünk kell a felhasználókkal, hiszen a tényleges használatot megcélzó, valós igényeknek megfelelő rendszert kell kifejlesztenünk. Az igények felmérése során olyan dokumentumokat kell tehát előállítanunk, amely a felhasználók számára is érthető és értelmezhető, így a felhasználók fogalmait és nyelvezetét kell alkalmaznunk, illetve tartózkodnunk kell az informatikai zsargonok és technikák említésétől.

Kiindulópontok

Nehézsége miatt, a követelmények tisztázása esetén célszerű meghatározni az üzleti modellezésen túl is néhány további kiindulópontot, melyek a tényleges munkafolyamat bevezető lépéseként fognak szolgálni. A későbbiekben ezek a bevezető lépések szükség szerint összeolvaszthatók a munkafolyamat tevékenységeivel. A bevezető lépések is egy-egy adott dokumentummal — annak létrehozásával vagy módosításával — kapcsolatos tevékenységek.

Előzetes követelmények felsorolása

Fejlesztése során a rendszerrel kapcsolatosan rendkívül sok elképzelés születik, melyek részben az adott iterációban megvalósításra kerülnek, részben valamely későbbi változatban tervezzük a beépítését, esetleg elvetjük annak kifejlesztését. Az előzetes követelmények felsorolása `(list candidate requirements)` folyamán ezeket az

elképzeléseket, „lehetséges követelményeket” gyűjtjük össze. A lista újabb elképzelések felvételével bővülhet, majd csökkenhet a mérete, ahogy a kiválasztott elemeket követelményeknek tekintjük és más produktumokká, pl. használati esetekké dolgozzuk át.

Minden elképzelésnek, lehetséges követelménynek adjunk egy rövid elnevezést és ahhoz egy néhány bekezdésnyi rövid magyarázatot. Mindezt célszerű kiegészíteni néhány, a megvalósítással, illetve annak tervezésével kapcsolatos információval, pl.

- státusz (ajánlott, ellenőrzött, stb.),
- várható költség (erőforrás-típus és a kifejlesztés ideje),
- prioritása (pl. kritikus, lényeges, lényegtelen),
- megvalósítás kockázati tényezője (pl. kockázatos, ismeretlen technológia alkalmazása miatt)…

Funkcionális követelmények rögzítése

A követelmények összegyűjtése során a megvalósításra kiválasztott elképzeléseket használati esetekként fogalmazzuk meg. Minden használati eset a rendszer egy adott használati módját, „funkcióját” reprezentálja. A használati esetek segítségével a követelményeket egzaktabb, egyben áttekinthetőbb módon rögzíthetjük.

Nem-funkcionális követelmények rögzítése

A nem-funkcionális követelmények a rendszer jellemzőit határozzák meg, mely lehet például a megvalósítási környezettel (platformmal) kapcsolatos követelmény, vagy vonatkozhat a rendszer kiterjeszthetőségére, megbízhatóságára, hatékonyságára, stb. A hatékonysággal kapcsolatos követelmények általában valamely funkcióra vonatkoznak, mely lehet például a válaszidőre vonatkozó korlátozás.

Ha a nem-funkcionális követelmény csak egyetlen használati esetre vonatkozik, akkor az magánál a használati esetnél is megadható.

Munkatársak

A követelmények összegyűjtése több „munkatársat”, azaz többfajta, speciális szaktudású személyt igényel. Ennek a munkafolyamatnak a specialitása, hogy a tevékenységeket a felhasználókkal való szoros együttműködésben, folytonos közvetlen vagy közvetett egyeztetéssel kell végrehajtani.

A *rendszer-elemző* `(system analyst)` felelőssége a használati esetekként modellezett követelmények összeállítása és kezelése, melyek együttesen jelentik a funkcionális és nem-funkcionális követelményeket. A rendszer-elemző feladata a rendszer határainak megrajzolása az aktorok és használati esetek eszközkészletével. A kialakuló használati eset modellnek teljesnek és konzisztensnek kell lenni. A fogalmak tisztázása és egyértelműsége a rövid fogalom-értelmezéseket tartalmazó *szójegyzék* `(glossary)` összeállításával biztosítható.

A rendszer-elemző feladata a használati eset modellnek, mint egésznek a kezelése. Az egyes használati esetek részletezése már más munkatárs feladata.

A *használati eset elemző* `(use case specifier)` feladata az egyes használati esetek részletezése, melyet a felhasználókkal történő folytonos egyeztetések során kell megvalósítania.

A *felhasználói-felület tervező* `(user interface designer)` alakítja ki a felhasználói felületek külső megjelenési formáját, mely gyakran a felületek előzetes változatainak, prototípusainak kifejlesztését is jelenti.

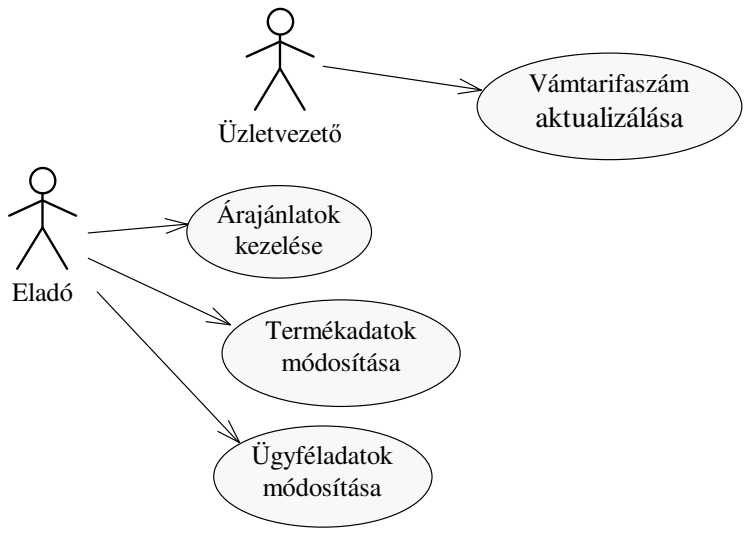
Az *architektúra tervező* `(architect)` a használati eset modell architektúrális nézetét állítja össze, amely a lényeges használati eseteket tartalmazza, így a fejlesztés ütemezésének is a kiindulópontja lesz.

Munkafolyamat

Aktorok és használati esetek keresése

Az aktorok és használati esetek keresésének célja a következő:

- a rendszer körvonalainak a megrajzolása, hogy egyértelműsítsük azt, hogy mi és hogy mi *nem* fog a rendszerhez tartozni,
- meghatározni, hogy ki és mi áll kapcsolatban a rendszerrel (aktorok) és milyen funkcionalitást várunk el a rendszertől (használati esetek),
- a gyakran használt fogalmak alapján szójegyzék készítése, mely alapvető szerepű lesz a használati esetek részletezése során.



8. Ábra: Használati eset diagram

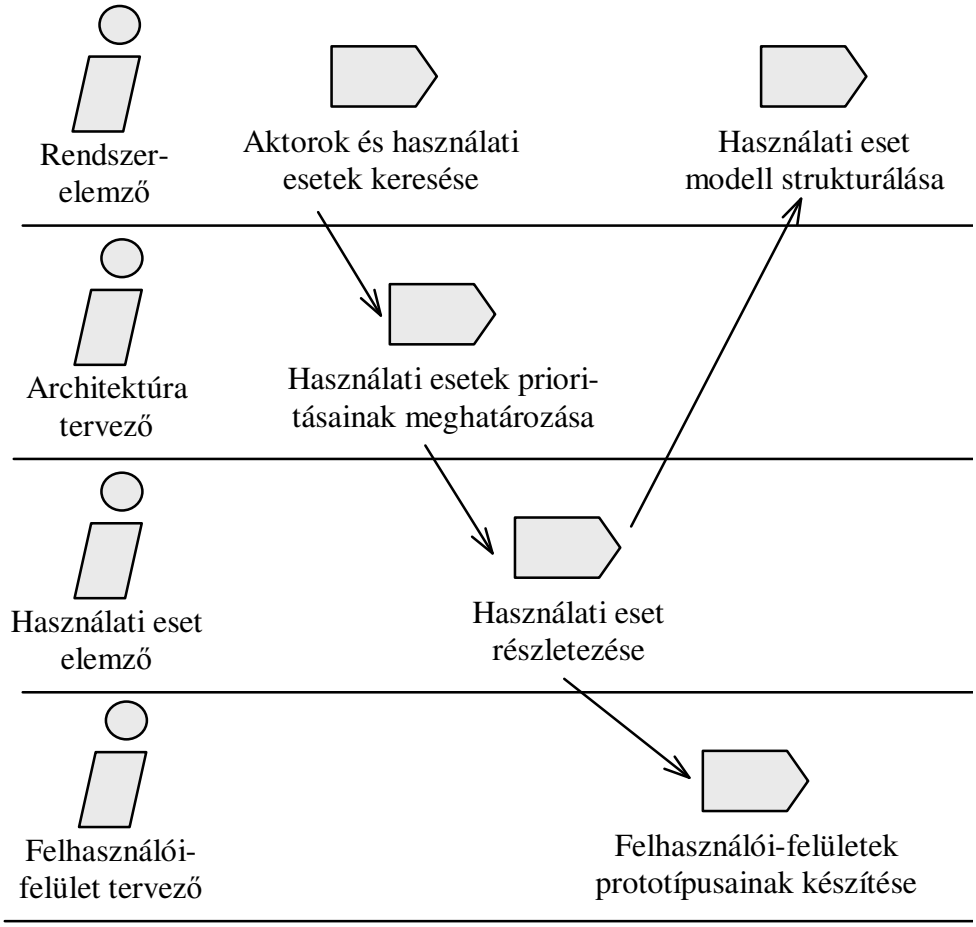
Az első lépés, az *aktorok keresése* során először összegyűjtjük azokat a szerepköröket, amelyeken keresztül személyek, illetve külső rendszerek a kifejlesztendő rendszerrel kapcsolatban állnak. A lehetséges aktorokat célszerű ellenőrizni, hogy találunk-e legalább egy konkrét felhasználót (ill. rendszert), aki ebben a szerepkörben fog megjelenni. Az aktorokat egy rövid névvel és egy összefoglaló leírással kell ellátni, melyben ki kell emelni

azt, hogy az aktor milyen szolgáltatásokat igényel a rendszertől, ill. melyek a felelősségei a rendszer használata során.

Az aktorok segítségével *meghatározzuk a használati eseteket*. Ehhez sorra veszünk minden aktort és felsoroljuk, hogy azok milyen funkciókon, azaz használati eseteken keresztül kapcsolódnak a rendszerhez és azokat mind ellátjuk egy-egy rövid elnevezéssel. Az aktorok például informálhatják a rendszert bizonyos külső változásokról, illetve a rendszer informálhatja az aktort valamely lényeges eseményről. A rendszer indítása, leállítása és kezelése ugyancsak funkciókként jelenik meg. A lehetséges használati esetek ellenőrzésére szolgál a „hasznos eredmény" módszere, mely szerint minden használati esetnek olyan eszköznek kell lenni, mellyel az aktor valamely célját érheti el.

A jelöltek közül a tényleges használati esetek kiválogatása után *röviden leírunk minden használati esetet*. A rövid leírás általában néhány mondatot jelent, de tartalmazhatja a funkció főbb lépéseinek a leírását is.

A *használati eset modell egészében történő leírásával* fejezzük be a tevékenységet, amikor diagramon ábrázoljuk, illetve szövegesen összefoglaljuk az aktorok és használati esetek viszonyait.



9. Ábra: Követelmények

Használati esetek prioritásainak meghatározása

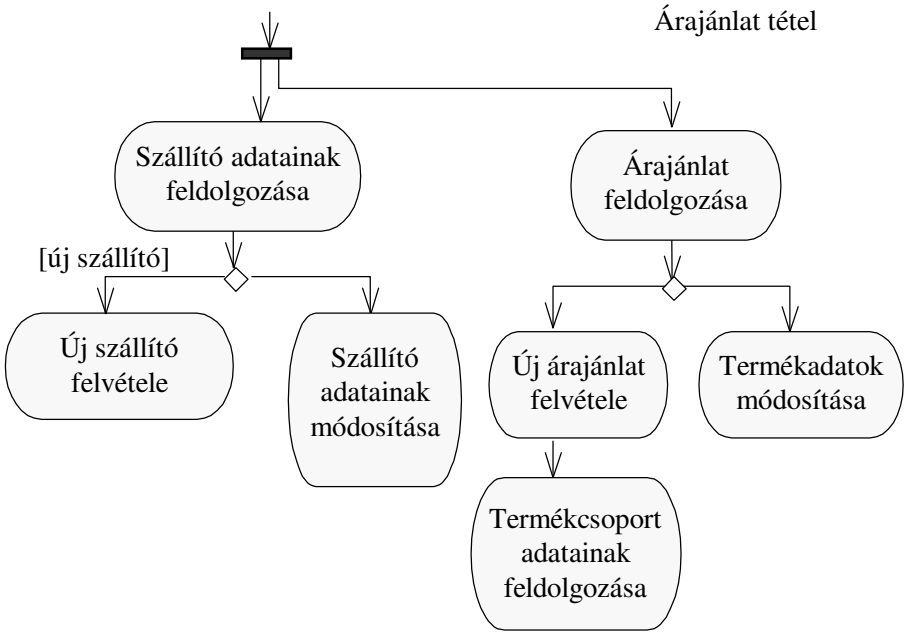
A használati eset modell összeállítása után a következő tevékenység a használati esetek prioritásainak meghatározása, azaz annak megállapítása, hogy mely eseteket kell a korai

iterációkban kifejleszteni és melyek megvalósítása halasztható a későbbi iterációkra. A prioritások („elsődlegességek") meghatározása a használati eset modellnek egy architekturális nézeteként jelenik meg, mivel kiemeljük a lényegesnek tartott, illetve megjelöljük a kiegészítőként tekintett használati eseteket.

Érdemes megjegyezni, hogy a prioritások meghatározásában jelentős szerepet kapnak nem-technikai szempontok is, például gyakoriak az üzletpolitikai döntések.

Használati eset részletezése

Az architekturálisan lényeges használati esetek kiválasztása után a következő feladat ezen esetek részletes leírása. A leírásnak tartalmaznia kell a funkció elindulásának és befejeződésének körülményeit, valamint a felhasználóval történő interakcióknak az ún. eseményfolyamát *(flow of events)*.



10. Ábra: Aktivitás-diagram

Az eseményfolyamnak lépésről lépésre haladva le kell írnia a felhasználóval történő kommunikációs lépéseket. Az eseményfolyamnak általában létezik egy normál *(„happy day")* lefolyása, melyet célszerű első lépésként felvázolni. A normál lépéssorozat mellett azonban sorra meg kell vizsgálni, hogy milyen esetekben, illetve feltételek mellett történhet a használati esetnek más lefolyása. Az alternatív utak, illetve hibás esetek több okból is bekövetkezhetnek, pl.

- a felhasználó a normáltól eltérő lépéssorozatot választ,
- valamely más aktor kihat az eseménysorozatra,
- az aktor helyetelen bemeneti értékeket ad meg,
- valamely rendszer-erőforrás hibásan működik, mely a teljes rendszer hibás működéséhez vezethet.

Az alternatív és hibás utak mellett célszerű feljegyezni olyan eseménysorozatokat is, melyek helyes működés esetén *nem* következhetnek be.

A használati esetek részletezését célszerű valamely félig formális, esetleg formális eszközzel megadni. A használati eset kezdetére és befejeződésére megadhatunk például (általában) szöveges elő- és utófeltételeket. A normál és alternatív eseményfolyam ábrázolására kiváló eszköz az UML állapot- vagy aktivitás-, esetleg valamely interakció-diagramja.

Felhasználói-felületek prototípusainak készítése

A használati esetek eseményfolyamjainak leírása után célszerű elkészíteni a felhasználói felületek prototípusait, azaz az aktorok és a rendszer közötti kommunikáció eszközeit. A felület-prototípusok különösen alkalmasak a felhasználókkal történő egyeztetésekre („hogyan lehet megadni egy bizonyos értéket"), mivel megjelenítik a rendszernek azt a külső képét, mellyel a felhasználó a későbbiekben is kapcsolatba kerül.

Első lépésként célszerű a felület-prototípusoknak csak a logikai szerkezetét vázolni, mellyel ellenőrizhetjük, hogy az alkalmas-e a használati eset által megkövetelt kommunikációra és tartalmazza-e az összes szükséges adatot és vezérlő elemet.

A vázlatok alapján ezután elkészíthetők a felhasználói felületek prototípusai. Ennél a lépésnél közvetlen programozás helyett célszerű felhasználói-felület tervező eszközt használni — ha az elérhető.

Használati eset modell strukturálása

A használati eset modell strukturálása a modell elemei (aktorok, ill. használati esetek) közötti viszonyok meghatározását, illetve a modell átrendezését jelenti. Az átrendezés célja elsősorban a redundancia csökkentése, optimális esetben pedig a kiküszöbölése.

A modell átrendezése történhet a *közös funkcionalitás kiemelésével*. A használati esetek részletezésekor megfigyelhetjük, hogy bizonyos esemény-sorozatok több használati esetben is megismétlődnek. Ekkor az ismétlődő műveletek kiemelhetők egy önálló használati esetbe, melyet az eredeti esetekhez include sztereotípiával jelölve kapcsolunk. A kiemelés történhet úgy is, hogy megfigyeljük, hogy bizonyos használati esetek valamely absztrakt műveletnek a konkrét változatai. Ezt a viszonyt általánosításként jelölhetjük, azaz a két használati esetet háromszögben végződő vonallal kapcsoljuk össze.

A használati esetek között megadhatunk még extend („kiterjeszt") sztereotípiájú kapcsolatot a *kiegészítő és opcionális funkció jelölése* esetén. Az extend módon kapcsolódó használati eset műveletei az alap használati eset eseménysorozatának adott (ún. kiterjesztési) pontján, adott feltételek mellett hajtódnak végre.

Összefoglalás

A követelmények rögzítése során azt határozzuk meg, hogy pontosan mit is kell kifejlesztenünk, mely a rendszer határainak meghúzását is jelenti. Kiindulópontként használhatjuk az előzetes követelmények felsorolását, valamint a funkcionális és nem-funkcionális követelményeket.

A követelményeket használati eset modellben adjuk meg, majd a lényeges használati esetek kiválasztásával azokat eseményfolyamokként részletezzük. Ezután elkészítjük a felhasználói felületek prototípusait, majd strukturálással áttekinthetőbbé teszzük a modellünk.

Kérdések

- Mi a követelmények rögzítésének a célja?
- Mi okozza ennek a lépésnek a nehézségeit?
- Milyen kiindulópontokat használhatunk a követelmények összegyűjtése során.
- A munkafolyamat milyen speciális igényként jelenik meg a munkatársak felé?
- Milyen lépésekben készítjük el a használati eset modell vázlatát?
- Hogyan részletezhetünk egy használati esetet?
- Mi alapján kereshetünk a normálistól eltérő eseménysorozatokat?
- Milyen elveket kövessünk a felhasználói felületek prototípusainak elkészítésénél?
- Hogyan strukturálható a használati eset modell?

Elemzés

Az összegyűjtött és rögzített követelményekből indul ki a fejlesztés következő munkafolyamata, az *elemzés* `(analysis)`. A követelmények rögzítésekor a *felhasználó* szempontjából írjuk le a rendszert, annak egy *külső* nézetét megrajzolva. Ezzel szemben az elemzés folyamán már a *fejlesztők* nyelvezetét használjuk és a rendszer alapvető *belső* felépítését határozzuk meg. A követelmény-rögzítés használati eset modelljének a felhasználókkal történő egyeztetés a célja. Az elemzés műveletével a rendszer belső, kezdeti formáját alakítjuk ki.

Az elemzés munkafolyamata során kialakuló modell célját a következő pontokban foglalhatjuk össze:

- a követelményeknek egy pontosabb leírását adja,
- a fejlesztők számára, az ő nyelvezetükön készül, így nagyobb formalitást és egzaktságot biztosít,
- a követelményekből következő alapstruktúrát rajzolja meg,
- kiindulópontja az ezt követő munkafolyamatnak, a tervezésnek.

Elemzési modell

Az elemzés eredménye az *elemzési modell*, ami egy fogalmi („koncepcionális”) objektum modellen alapszik, amelyben a követelményeknél megjelenő fogalmak osztályokként szerepelnek. Az elemzési modell segítségével a követelményeket pontosabban és adott formalizmus jelöléseivel fogalmazzuk meg. Az elemzési modell azonban a rendszernek csak az absztrakt képe, így nem tartalmazza a nem-funkcionális követelmények megoldását, sem a tervezés során kiválasztandó programozási nyelvvel, platformmal, operációs rendszerrel, felhasznált keretrendszerekkel, stb. kapcsolatos döntéseket. A Unified Process összeállítói az elemzési és a tervezési modell részletességének a viszonyát nagy átlagban 1:5 arányra becsülik.

Az elemzési modellt osztálydiagramokkal megadott objektum-modell köré szervezzük. Az osztálydiagramok főszereplői az *osztályok*, melyeket a Unified Process az elemzés során három fő csoportba sorol, melyek külön sztereotípiákkal és ahhoz rendelt külön ikonokkal jelöl.

A `boundary` sztereotípiával megadott *határ-osztályok* `(boundary class)` szerepe az aktorok és a rendszer közötti kommunikáció biztosítása, így ezek az osztályok a rendszer peremét jelölik. A határ osztályok általában felhasználói felületekként (ablakok, panelek, nyomtatási képek…) vagy más kapcsolódási elemként (szenzorok, stb.), esetleg külső rendszer interfészeként jelennek meg. A határ-osztályok leírásánál az adatcserére célszerű összpontosítani, az interakciók fizikai megjelenését csak későbbi munkafolyamatok során kell meghatároznunk.

Az entity sztereotípiával megadott *entitás-osztályok* `(entity class)` általában a valóságban is megjelenő fogalmakat reprezentálnak, melyeket a rendszer tárol („perzisztencia”). Az entitás-osztályok együttesen a rendszer alap-adatszerkezetét adják meg, a rendszer belső információs közegét.

A control sztereotípiájú *vezérlő osztályok* `(control class)` egy adott használati esethez kapcsolódó vezérlést, tevékenységsorozatokat, tranzakciókat valósítanak meg,

olyan üzleti folyamatokat és műveleteket, melyek nem kapcsolhatók más entitás-osztályhoz.

Az elemzési modell osztálydiagramjain a három típusnak megfelelő osztályok, azok attribútumai és a hozzájuk közvetlenül kapcsolható műveletek, valamint az osztályok közötti kapcsolatok és egyéb (pl. általánosítás-pontosítás) viszonyok szerepelnek.

A modell hierarchikus tagolására szolgálnak az *elemzési csomagok (analysis package)*. Egy csomagon belül elhelyezkedő osztályoknak és belső csomagoknak egymáshoz szorosan kell kapcsolódni, a külső elemekhez történő kapcsolódást pedig a minimumra kell csökkenteni. A Unified Process megkülönbözteti az ún. *szolgáltató csomagokat (service package)*, melyek funkciói nem kötődnek közvetlenül egy konkrét használati esethez, hanem a fejlesztendő rendszernél általánosabb megoldási módokat gyűjtenek egybe (pl. fax-küldő rendszer).

Az elemzési modellben szerepelnie kell az egyes használati esetek elemzési szintű megvalósításainak *(use case realization)*, mely a használati esetben megjelenő osztályok közötti egyes interakció-sorozatokat ábrázolják, általában valamely interakció-diagram eszközeivel.

Elemzési szintű architektúra leírás

Az elemzés egy összefoglaló dokumentuma az elemzési szintű architektúra leírás, melyben a leglényegesebb modell-elemeket foglaljuk össze, sorrendben a következőket:

- felső-szintű (és az architektúra szempontjából lényeges) elemzési csomagokat és a közöttük lévő függőségeket,
- alapvető fontosságú elemzési osztályokat, azok leglényegesebb attribútumait, műveleteit és egymáshoz való viszonyait,
- lényeges és kritikus használati esetek megvalósításait, melyeket a használati eset modell architekturális nézetébe már kiemeltünk.

Munkatársak

Az elemzés során az *architektúra tervező (architect)* feladata az elemzési modell egészére a korrekt, konzisztens és értelmezhető forma biztosítása. Az architektúra tervező a felelős mindazon modellelemek kidolgozásáért és kezeléséért, amelyek helyet kapnak az architektúra leírásban.

A *használati eset mérnök (use-case engineer)* feladata az elemzés során egy vagy több használati eset elemzési szintű megvalósításának a kidolgozása, úgy, hogy teljesítse az esettel kapcsolatos követelményeket.

A *komponens mérnök (component engineer)* feladata egy vagy több osztály és csomag kidolgozása és kezelése, osztályok esetén beleértve az attribútumokat, műveleteket és a más osztállyal való viszonyokat.

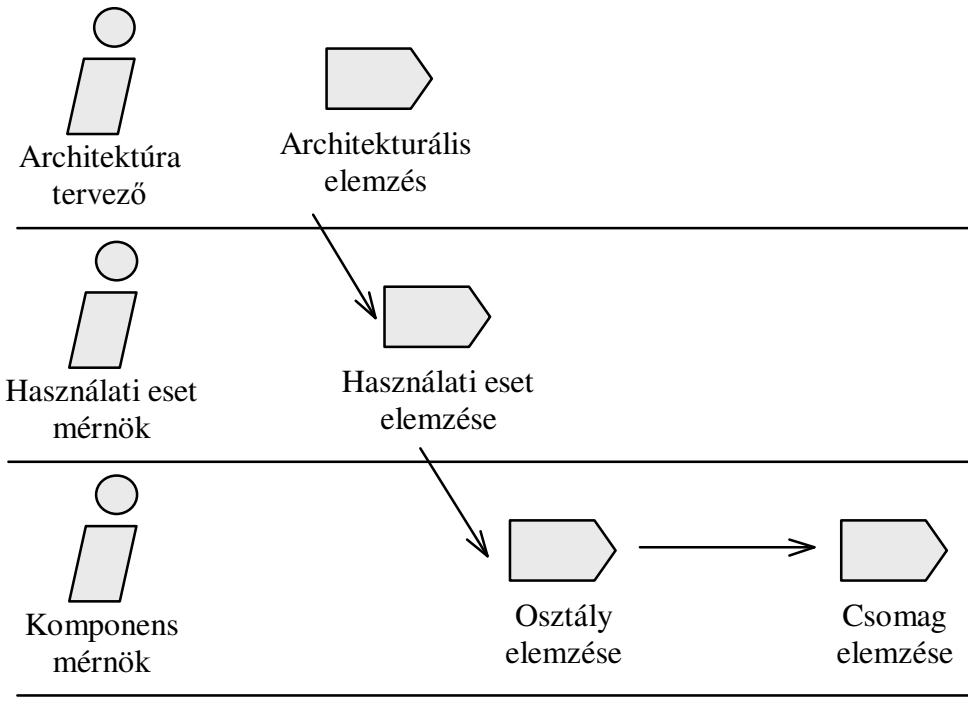
Munkafolyamat

Architekturális elemzés

Az architekturális elemzés során *meghatározzuk* a felső szintű *elemzési csomagokat* `(analysis package)`, így a modellt kisebb, könnyebben kezelhető részekre bontjuk fel. Ez történhet az elemzés első lépéseként is, de gyakori, hogy munka közben, nagyszámú modellelem áttekinthetőbbé tétele miatt hajtjuk végre ezt a műveletet. A csomagokra történő felbontást segíti a követelmények funkció-csoportjainak a meghatározása, de érdemes megjegyezni, hogy egy használati eset általában több csomag együttes működésével valósítható meg. A rendszer konkrét működéséhez közvetve kapcsolódó, adott jellegű, általános szolgáltatást nyújtó csomagokat ábrázoljuk *szolgáltató-csomagokként*. Az elemzési csomagok műveleteik megvalósításához felhasználhatják más csomagok műveleteit, mely viszonyt a csomagok közötti függőségként jelöljük.

Az architekturális elemzés során *azonosítjuk a nyilvánvaló entitás osztályokat*. Ekkor kiválasztjuk az összegyűjtött követelményekből (szakterületi modellből vagy az üzleti modellből) a legfontosabb szerepet játszó (kb. 10-20) osztályt. A használati eset megvalósítása esetén még további osztályok is megjelenhetnek, ezért célszerű csak a központi szerepű osztályokat kiemelni, így ezek között nem szerepelhet például olyan, amely egyetlen használati esetben sem kap szerepet.

Az architektúra tervező feladata a több modell-elemre vonatkozó, *közös speciális követelmények meghatározása*, melyek a későbbi tervezés és implementáció során alapvető szerepet kapnak. Ilyen követelmények például a tárolás („perzisztencia”), osztottság és konkurencia kezelése, biztonság, hibatűrés, tranzakció-kezelés, stb. kérdései. A figyelembe veendő speciális követelményekre a Unified Process elemzési mintákat ajánl, például tárolás esetén a következő szempontokat kell számszerűsíteni: tárolandó objektumok mérete, száma, tárolási idő, módosítási gyakoriság és megbízhatóság.



11. Ábra: Elemzés

Használati eset elemzés

A használati eset elemzés *(use-case analysis)*, vagy más szóval a használati eset részletezés *(use-case refinement)* kezdetén *azonosítjuk az elemzési osztályokat*, melyeket határ-, entitás- vagy vezérlő-osztályokként adunk meg.

Az elemzési osztályok keresésére a Unified Process a következő javaslatokat adja:

- Az entitás-osztályokat a használati esetek leírása alapján azonosíthatjuk. Vizsgáljuk meg, hogy a használati eset megvalósítása milyen adatokat érint és kezel. Az entitás-osztályok keresését jelentősen megkönnyíti a szakterületi modell.
- Minden felhasználónak határozzunk meg egy alapvető határ-osztályt, amely központi szerepű a rendszer és a kiválasztott felhasználó közötti kommunikáció során. Ez a határ-osztály lehet például a felhasználó számára megjelenő kiindulási képernyő.
- Minden entitás osztály esetén határozzunk meg egy olyan határ-osztályt, amely lehetővé teszi az entitás-osztályra vonatkozó információcserét.
- Minden külső rendszer esetén határozzunk meg olyan határ-osztályt, amely a kommunikációs felületet reprezentálja.
- Használati esetenként vizsgáljuk meg, hogy szükséges-e olyan vezérlő-osztály, amely koordinálja a használati eset működését. Jegyezzük meg, hogy egyszerűbb esetekben a vezérlés természetesebben megadható a határ-osztályokban.

Az elemzési osztályok megkeresése után a következő feladat az *elemzési objektum interakciók leírása*, mely a használati eset esemény-sorozatának pontos és formalizált részletezése. Az interakciót együttműködési diagrammal *(collaboration diagram)* ábrázoljuk, mely elkészítésére a Unified Process a következő lépéseket ajánlja:

- A használati eset indítását egy konkrét aktortól egy határ-objektumig vezető üzenet váltja ki.
- Az azonosított elemzési osztályoknak legalább egy objektum felel meg, különben az osztály felesleges.
- A használati eset működését üzenetekkel írjuk le, az üzenetek azonban általában nem konkrét műveletet jelölnek, hanem csak a végrehajtás logikai felbontását.
- Az üzenetek az objektumok közötti kapcsolatokon terjednek, ezek a kapcsolatok azonban nem feltétlenül utalnak asszociációkra.
- Az elemzés során nem a pontos végrehajtási sorrend megadására összpontosítunk, hanem a hangsúly az objektumok közötti viszonyon és azokkal szemben támasztott követelményeken van.
- A használati eset modellben ábrázolt, használati esetek közötti (include, extend vagy általánosítás) viszonynak az együttműködési diagramon is meg kell jelenni.

Utolsó lépésként *összegyűjtjük a* használati esettel kapcsolatos *speciális követelményeket*, így a nem-funkcionális követelményeket, amelyeket figyelembe kell venni a tervezés és implementáció során.

Osztály elemzése

A használati eset elemzés tevékenysége során már azonosítottuk az elemzési osztályokat. A következő tevékenység, az osztály elemzésének végrehajtásakor minden egyes osztályra részletesen is kifejtjük azok feladatkörét és egymáshoz való viszonyait.

Először *azonosítjuk* az osztály *felelősségeit*, melyet úgy kaphatunk meg, hogy sorra vesszük az összes szerepet, amelyben az a különböző használati esetekben megjelenik.

Ezután *azonosítjuk* az osztály *attribútumait*, amelyek az osztály egy-egy jellemzőjét reprezentálják. Az attribútumok a felelősségek alapján könnyen kikereshetők és megadásukkor célszerű betartani a következő szabályokat:

- az attribútum neve legyen főnév,
- az elemzés során koncepcionális típusokat használjunk, ne pedig az implementációs környezet típusait (pl. „mennyiség”, ne pedig: int),
- konzisztens típusokat alkalmazzunk,
- ha az attribútumok önállósággal rendelkeznek, akkor azokat egy újabb osztályba kell kiemelnünk,
- ha az attribútumok miatt egy osztály túlságosan bonyolulttá válik, akkor azt lehet, hogy célszerű több osztályra szétbontani,
- entitás-osztályok attribútumai általában a szakterület alapján könnyen azonosíthatók,
- határ-osztályok attribútumai általában az aktor által kezelt információkat jelölik,
- a vezérlő-osztályok attribútumai a művelet végrehajtását kísérik végig.

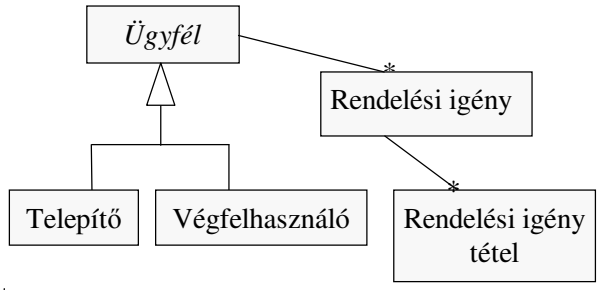
Példánk alapján következzen most néhány osztály elemzési szintű attribútuma:

- Ügyfél: név, cím, telefonszám, fax szám
- Telepítő: Ügyfél AZ, bankszámla szám, adószám, ügyintéző, adós jelző
- Végfelhasználó:
- Rendelési igény: rendelés felvételének dátuma, előleg összege
- Rendelési igény tétel: termék mennyiségi egysége, megrendelendő termék mennyisége,
- tétel megrendelésének dátuma, kiértesítés dátuma

Az attribútumok után *azonosítsuk* az osztályok közötti *asszociációs és aggregációs viszonyokat*. Az objektumok interakciója az objektumok közötti kapcsolatok mentén zajlik le, mely kapcsolatok gyakran az objektumoknak megfelelő osztályok közötti asszociációk konkrét példányai. A komponens mérnök a kapcsolatokat vizsgálva megállapítja, hogy melyeknek kell az osztályok közötti asszociációkként is megjelenni, illetve — az élettartamok összefüggései alapján — melyek igényelnek aggregációs kapcsolatot. Az asszociációk és aggregációk meghatározásakor meg kell adni azok számosságát („multiplicitását”), elnevezésüket, a szerepkörök neveit, illetve szükség esetén az asszociációs osztályokat, a szerepkörök rendezettségét, illetve a minősítőket `(qualifier)`. Célszerű aggregációt alkalmaznunk, ha

- a fogalmak között (amelyeket az osztályok reprezentálnak) fizikailag is tartalmazási viszony van (pl. az előadóteremben az előadó és a hallgatók),
- a fogalmak között kompozíciós viszony van (a szobák a lakás alkotóelemei),
- a fogalom az elemek logikai gyűjtőfogalma (az osztály és a tanulói).

Ezután az összegyűjtött attribútumok és asszociációk segítségével *meghatározzuk* az *általánosításokat*, az elemzés során azonban csak a fogalmi szintű általánosítás-pontosítás viszonyokat jegyezzük fel, melyek az elemzési modell megértését könnyítik.



12. Ábra: Osztálydiagram

A tevékenység befejezéseként minden egyes osztályra összegyűjtjük és *megadjuk a speciális követelményeket*.

Csomag elemzése

Az elemzés befejezéseként egyenként sorra vesszük és elemezzük a csomagokat, a következő szempontokra összpontosítva:

- biztosítjuk, hogy az elemzési csomag a többi csomaghoz a lehető legkisebb mértékben kapcsolódik,
- biztosítjuk, hogy a csomag alkalmas bizonyos szakterületi osztályok, vagy használati esetek egy csoportjának a reprezentálására,
- megadjuk a csomagok közötti függőségeket, így megbecsülhetjük a későbbi változtatások hatásait.

A célok úgy érhetők el, ha ellenőrizzük, hogy a csomagok csak a szorosan összekapcsolódó, funkcionálisan összefüggő osztályokat tartalmazzák, és a lehető legkisebbre korlátozzuk a csomagok közötti kapcsolódásokat.

Összefoglalás

A követelmények rögzítésével ellentétben, amikor felhasználói szemmel írjuk le a rendszert, az elemzés már a fejlesztők szempontját képviseli, és a rendszer belső felépítését határozza meg. Az elemzés a rendszer belső, kezdeti formáját alakítja ki, egy formálisabb és egzaktabb nyelvezetet használva.

Az elemzés eredménye az elemzési modell, mely alapvetően egy osztálydiagramokként megadott fogalmi objektum-modellen alapszik. Az osztályokat sztereotípiákkal három csoportba soroljuk: a határ-osztályokkal a rendszerrel történő kommunikációt biztosítjuk, az entitás-osztályok általában a valóságban is megjelenő fogalmakat reprezentálnak, a vezérlő osztályok pedig egy összetett tevékenységsorozatot kísérnek végig. Az elemzés másik produktuma az összefoglaló szerepű elemzési szintű architektúra leírás, melyben a leglényegesebb modell-elemeket foglaljuk össze.

Az elemzési modellt alapvetően a használati esetek elemzési szintű megvalósításai alapján állítjuk össze. Az osztállyal kapcsolatos információkat az osztályt érintő interakciók vizsgálata alapján válogatjuk össze.

Kérdések

- Milyen szempontból készül el a követelmények rögzítése és milyen szempontból az elemzés?
- Milyen célt szolgál az elemzési modell?
- Mi az elemzési modell központi eleme?
- Milyen arányra becsülik a RUP összeállítói az elemzés és tervezés költségviszonyát?
- Az elemzési szintű architektúra leírásban milyen elemek szerepelnek?
- Mely három csoportba soroljuk az elemzési osztályokat?
- Mit nevezünk szolgáltató csomagoknak?
- Hogyan kereshetjük meg az entitás-osztályokat?
- Milyen lépésekkel írhatjuk le az objektum-interakciókat?
- Milyen szabályokat alkalmazunk az attribútumok megadásakor?

Tervezés

A tervezés során az elemzés által meghatározott keretek töltjük fel tartalommal és formáljuk a rendszert teljes alakká, amely teljesíti a korábban meghatározott összes funkcionális és nem-funkcionális követelményeket.

A Unified Process a tervezés célját a következő főbb pontokban foglalja össze:

- A tervezésnek a készítendő rendszerről teljes mélységében részletes képet kell adnia, a kiválasztott programozási nyelvekre, operációs rendszerre, technológiákra vonatkozóan is.
- Össze kell állítania az implementációhoz szükséges dokumentumokat.
- Az implementáció ütemezéséhez a rendszert kezelhető részekre kell bontania.
- Meg kell határoznia a különböző részek közötti interfészeket.
- Adott jelölési rendszert felhasználva a tervezési döntéseket egzaktul meg kell adnia.
- Meg kell rajzolni a rendszer implementációjának a vázát.

Az elemzéssel összehasonlítva, a tervezés nem a fogalmi modell kialakítására törekszik, hanem a rendszernek az annál konkrétabb fizikai modelljét határozza meg, amely az implementáció „alaprajza” *(blueprint)* lesz. Az elemzés modellje még több tervezési módon is konkretizálható, a tervezés azonban már konkrét platformokra és technológiákra épít. Az „alaprajznak” már formálisnak és pontosnak kell lenni, hogy az alapján az implementáló programozók további bizonytalanság nélkül felépíthessék a teljes rendszert. A Unified Process átlagban a tervezés költségeit az elemzési költség ötszörösére becsüli.

Tervezési modell

A tervezés munkafolyamatának a legfontosabb eredménye a *tervezési modell* *(design modell)*, ami egy olyan objektum modell, amely a használati esetek fizikai megvalósítási módját írja le, figyelembe véve az összes funkcionális és nem-funkcionális követelményt, valamint a megvalósítási környezetet. A tervezési modell tervezési rendszerekből és alrendszerekből áll, melyek tervezési osztályokat, a használati esetek tervezési szintű megvalósításait és interfészeket tartalmaznak.

A tervezési modell az implementációs modell absztrakciója, „alaprajza”, ami azt jelenti, hogy a megfelelő leképezésekkel *(mapping)* megkaphatjuk az implementációs dokumentumokat, illetve, hogy a tervezési modell a megvalósított rendszer áttekintő terveként szolgál.

A tervezési modellben alapvető szerepűek a *tervezési osztályok* *(design class)*, amely az implementációs osztályok közvetlen absztrakciója. A tervezési osztályok jellemzői a következők:

- A tervezési osztály leírásakor a kiválasztott programozási nyelv fogalmait és típusait használjuk.

- Általában megadjuk a jellegzetességek láthatóságát `(visibility)`, pl. C++ nyelv esetén ezek a private, protected és public kulcsszavakkal jelölik.
- Az osztályok közötti viszonyok közvetlen implementációs technikákra utalnak. Például az általánosítás általában örökléskén fog megjelenni. Az alapértelmezéstől való eltérések és a nyelvfüggő eszközök sztereotípiákkal adhatók meg.
- Az osztály műveletei az osztály metódusaiként jelennek meg az implementáció során.
- Tervezési osztály esetén sztereotípiával adhatjuk meg a kiválasztott implementációs elemet, pl. database, form.
- A tervezési osztály gyakran megvalósít valamely interfészt, így az osztály objektumai az adott interfésznek megfelelő módon kezelhetővé válnak.
- A tervezési osztály lehet aktív, azaz konkurensen működhet, más műveletekkel párhuzamosan. A párhuzamos működés szükségességét azonban célszerű a későbbiekben, az implementáció során meghatározni.

Az osztály-diagram kiegészítéseként a használati esetek tervezési szintű megvalósításakor az eseménysorozatokat interakció-diagramokkal, elsősorban pedig szekvencia-diagramokkal ábrázoljuk. A diagramoknak a konkrét interakciók absztrakcióját kell reprezentálni, így az üzeneteknek konkrét metódushívásoknak kell megfelelni.

A tervezési modell elemeit *tervezési rendszerek és alrendszerek* segítségével bontjuk szét kezelhető részekre. A tervezési rendszer tervezési osztályokat, használati esetek tervezési szintű megvalósítását, interfészeket és más tervezési alrendszereket tartalmaz. Az interfészek olyan eszközök, amelyeken keresztül (műveletekként) elérhető a rendszer funkcionalitása.

A tervezési rendszer

- tervezési modell elemek nagyobb egységeit jelenti,
- a megvalósítás nagyobb méretű egységeit jelöli,
- alkalmas valamely felhasznált önálló rendszer reprezentálására,

A tervezési rendszerek közül célszerű elkülöníteni és megjelölni a szolgáltató rendszereket, amelyek adott, a rendszerhez logikailag kevésbé kapcsolódó, részben függetleníthető funkciócsoportot valósítanak meg.

A tervezési modell egy lényeges, összefoglaló eleme a tervezési modell architekturális nézete, amely a modell leglényegesebb elemeit tartalmazza. Ezek az elemek általában a következők:

- a tervezési modell alrendszerekre történő bontása, azok interfészei és függőségei,
- a kulcsfontosságú tervezési osztályok, amelyek általában az architekturálisan lényeges elemzési osztályok tervezési szintű pontosításai, az aktív osztályok és más, központi szerepű osztályok,
- a lényeges és kritikus funkcionalitást megvalósító használati eset tervezési szintű megvalósításai.

Telepítési modell

A tervezés másik eredménye a telepítési modell *(deployment model)*, amely a rendszer részeinek az ún. csomópontokon (pl. számítógépeken) való fizikai elhelyezkedését ábrázolja.

A telepítési modellben:

- minden csomópont valamely számítógépet, vagy hasonló jellegű hardware elemet jelöl,
- a csomópontok közötti kapcsolatok a közöttük lévő kommunikáció eszközére utalnak, pl. Internet, intranet, soros vonal.
- több diagramon más konfigurációk is leírhatók, pl. teszt- vagy szimulációs konfigurációk,
- a csomópont funkcionalitását a csomóponton elhelyezett komponensekként jelöljük.

Fontosságánál és áttekintő jellegű szerepénél fogva a telepítési modellt célszerű teljes egészében befoglalni az architektúra leírásba.

Munkatársak

A tervezés alaptevékenységeiben az elemzéssel azonos munkatársak vesznek részt, azonban most a feladataik és felelősségeik a fejlesztés technikaibb területeire vonatkoznak. Mivel a feladatok némileg más jellegű, főleg a kiválasztott alkalmazási környezetre vonatkozó szaktudást is igényelhetnek, ezért lehetséges, hogy a fejlesztés során ezeket a tevékenységeket ténylegesen más konkrét személyek is végzik.

A tervezésben is az egyik legfontosabb munkatárs az *architektúra tervező (architect)*, aki a tervezési és telepítési modell egészéért, annak integritásáért, konzisztens és áttekinthető voltáért a felelős. Az architektúra tervező feladata még a tervezési és telepítési modell architekturális nézetének a kidolgozása és kezelése. Ezek a nézetek az architektúra leírás elemeiként jelennek meg.

A *használati eset mérnök (use-case engineer)* egy vagy több használati eset tervezési szintű megvalósításáért a felelős, biztosítva, hogy a kidolgozás teljesíti a funkcionális és nem-funkcionális követelményeket és megfelelően egzaktul leírja a működést, előkészítve az implementációt.

A *komponens mérnök (component engineer)* definiálja és karbantartja egy vagy több tervezési osztály attribútumait, műveleteit, kapcsolatait más osztályokkal, és az azzal kapcsolatos implementációs követelményeket. A komponens mérnök a felelős még egy vagy több alrendszer tartalmáért, valamint hogy azok interfészei és más rendszerrel való kapcsolatai megfelelők és a minimumra korlátozottak. A mérnök felelőssége kiterjed arra is, hogy az alrendszerek megfelelően valósítják-e meg a szükséges interfészeket.

Munkafolyamat

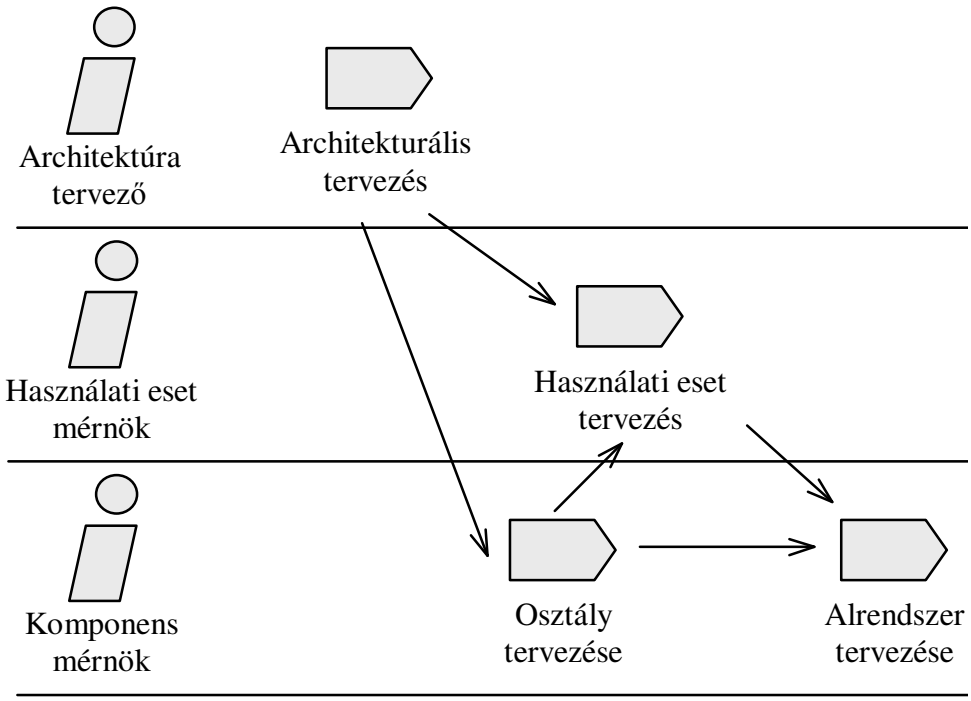
Architekturális tervezés

Az architekturális tervezést a *csomópontok és hálózati konfigurációk meghatározásával* kezdjük, mivel a hálózati konfiguráció általában közvetlenül kihat a software architektúrára, így például szükségessé teheti aktív (párhuzamosan működő) osztályok felvételét. A hálózati konfigurációt általában a három-rétegű modell *(three-tier)* alapján szervezzük meg, ahol az első réteg a felhasználói felületeket tartalmazó kliens-programot jelenti, a középső réteg az üzleti logikát megvalósító szerver-program, a nagymennyiségű adatok tárolása pedig a harmadik réteg adatbáziskezelőjével történik.

A hálózati konfiguráció meghatározásakor a következő szempontokra összpontosítunk:

- milyen csomópontokat igényel a rendszer és azokon milyen működési sebesség és memóriaméret szükséges,
- a csomópontok között milyen kapcsolat, azaz milyen kommunikációs lehetőség van,
- hogyan jellemezhető a kommunikációs-igény, ahhoz milyen sávszélesség, elérhetőség és biztonság szükséges,
- a biztonság érdekében szükséges-e a feldolgozások redundánssá tétele, mentések, hibatűrő módok, stb. alkalmazásával?

A különböző hálózati konfigurációkat, beleértve a teszt- és a szimulációs konfigurációkat külön telepítési diagramokon kell ábrázolni.



13. Ábra: Tervezés

A következő lépés az *alrendszerek és interfészeik azonosítása*. Az alrendszerekkel a tervezési modellt kezelhető egységekre bontjuk fel. A felbontás történhet a tervezés kezdetén, vagy munka közben, a rendszer bővülésével. A rendszert általában több csomagot is magába foglaló rétegekre `(layer)` bontjuk, ahol is egy réteg csomagjai csak egymásra vagy az alsóbb rétegekre hivatkoznak. A diagramokon célszerű ábrázolnunk az alrendszerek közötti függőségeket `(dependency)`, mivel így könnyebb a későbbi változtatások tervezése. Végül a függőségek alapján meghatározzuk, hogy az alrendszerek közötti kapcsolat milyen interfészeken keresztül valósul meg

Az *architekturálisan lényeges tervezési osztályok azonosításának* tevékenységét gyakran a tervezési munka kezdetén hajtjuk végre. A kiindulópontunk az architekturálisan lényeges *elemzési* osztályok, melyek általában tervezési osztályokként is megjelennek. Ezen kívül fel kell venni még a rendszer működéséhez szükséges aktív osztályokat is. Aktív osztályra szükség lehet például a következő okok miatt:

- a rendszer komponensei több csomóponton is elhelyezkedhetnek, a csomópontok közötti kommunikáció pedig csomópontonként legalább egy aktív osztályt igényel,
- a hatékonyság és több aktor párhuzamos kiszolgálása miatt gyakran szükségesek aktív vezérlő osztályok,
- a rendszer kezelésével kapcsolatos feladatok is igényelhetik aktív osztályok felvételét.

Az architekturális tervezés kiegészítéseként *általános tervezési mintákat alkalmazunk* a speciális, pl. nem-funkcionális követelmények teljesítésére. Az elemzés során már meghatároztuk a követelmények jellegét és számszerűsítettük az azokkal kapcsolatos értékeket. A tervezés során az elérhető eszközök, technológiák és módszerek közül kiválasztjuk azt, amely megfelel a követelményeknek. Gyakori követelmények például a következők: tárolás („perzisztencia”), osztottság és konkurencia kezelése, biztonság, hibatűrés, tranzakció-kezelés, stb. kérdései.

Használati eset tervezése

A használati eset tervező a tervezés során egyenként megvizsgál egy vagy több használati esetet és leírja azok tervezési szintű megvalósítását. Először *azonosítjuk a résztvevő tervezési osztályokat*.

- A használati eset elemzési szintű megvalósítása alapján megvizsgáljuk az abban szereplő elemzési osztályokat, majd megkeressük az azoknak megfelelő tervezési osztályokat.
- A használati esettel kapcsolatos speciális követelmények megvalósítása is igényelheti újabb tervezési osztályok felvételét.
- Ha a megvalósítás további osztályokat is igényel, akkor azokat az architektúra tervező vagy a komponens mérnök együttműködésével kell felvenni.

A használati eset megvalósításához leírjuk az objektum-interakciókat. A kiindulópontjaink a tervezési osztályok, valamint a használati eset elemzési szintű megvalósítása. Az interakció ábrázolására a Unified Process a szekvencia-diagramokat ajánlja, mely konkrét aktorokat, tervezési objektumokat és a közöttük lezajló üzenetváltásokat tartalmazza. Az interakciót egzaktul kell leírnunk, így az üzeneteknek

konkrét metódushívásoknak kell megfelelni. Az interakció alternatív működései a szekvencia-diagram feltételes üzenetküldéseiként ábrázolható, vagy összetettebb esetekben külön szekvencia-diagramok segítségével.

Esetenként a használati eset megvalósítását nem osztályok közötti interakciókkal célszerű leírnunk, hanem a résztvevő alrendszerekkel és használt interfészeikkel. Ehhez *azonosítanunk kell a résztvevő alrendszereket és interfészeket*.

A használati eset megvalósításakor össze kell gyűjtenünk, és fel kell jegyeznünk az esettel kapcsolatos *speciális implementációs követelményeket*, például a nem-funkcionális követelményeket. A tervezés munkafolyamata során ezeket a követelményeket csak feljegyezzük, figyelembe vételük és megvalósításuk az implementáció feladata.

Osztály tervezése

Az osztály tervezésekor kiemelünk és *körvonalazunk egy-egy tervezési osztályt*, megadva annak az alapvető kereteit.

- Határ-osztályok *(boundary class)* tervezése az alkalmazott felhasználói felület, illetve más illesztési felület technológiájára épül. A technológia kiválasztása így az osztályt alapszerkezetét is meghatározza.
- Tárolt („perzisztens”) információkat reprezentáló entitás-osztályok *(entity class)* alapszerkezetét a tárolási módszer határozza meg, így igényelheti például a relációs adatbázis táblájából történő betöltést, illetve mentést.
- A vezérlő osztályok *(control class)* alapszerkezetét a vezérlés jellege határozza meg, így általában figyelembe kell venni az osztottsággal, hatékonysággal, esetleg a szinkronizációval kapcsolatos problémákat. A vezérlő osztályok gyakran adott tranzakciót kísérnek végig, így kialakításuk a használt tranzakciós technológiát követi.

Az osztály *műveleteinek azonosításakor* a metódusok szignatúráját (nevét, paramétereit) és leírását a használt programozási nyelv terminológiájával és az ott érvényes szintaktikai szabályoknak megfelelően írjuk le. A műveletek keresésénél több kiindulópontot is használhatunk:

- Az elemzési osztálynál felsorolt felelősségek gyakran a tervezési osztály műveleteiként valósulnak meg.
- Az elemzési osztálynál meghatározott speciális követelményeknek (pl. adatbázisban tárolás) a tervezési osztályban is meg kell jelenni, általában az osztály általános keretét meghatározó technológiaként.
- Az elemzési osztálynál meghatározott interfészek műveleteivel a tervezési osztálynak rendelkeznie kell.
- Az osztálynak rendelkeznie kell a használati eset megvalósításához szükséges műveletekkel.

Az attribútumok azonosításakor ugyancsak a használt programozási nyelv szintaktikáját alkalmazzuk. Az attribútumok az osztály jellegzetességei, melyek a legtöbb esetben a műveletek végrehajtásához szükségesek. Érdemes követni a következő javaslatokat:

- Az elemzés során azonosított attribútumok a tervezés során egy vagy több attribútumként jelennek meg.
- Az attribútum típusának megadásakor a programozási nyelv eszközeit kell alkalmaznunk.
- Ha nagyszámú attribútumunk vagy viszonylagos függetlenséggel rendelkező attribútumcsoportunk van, akkor azokat célszerű kiemelni egy külön osztályba.

Az attribútumok mellett *azonosítanunk* kell az *asszociációs és aggregációs viszonyokat*. Az objektum-interakciók gyakran megkövetelik a megfelelő osztályok közötti asszociációkat. A komponens mérnök feladata a kapcsolatok vizsgálatával a szükséges asszociációk, illetve az aggregációs viszonyok bejelölése.

- Az elemzés során azonosított asszociációk és aggregációk a tervezés során egy vagy több asszociációs, illetve aggregációs viszonyként jelennek meg.
- A tervezéskor pontosítanunk kell az asszociációk számosságát, szerepköreinek neveit, asszociációs osztályait és minősítőit. A pontosításkor már a kiválasztott programozási nyelv jelöléseit és szintaktikai szabályait kell alkalmaznunk.
- A tervezés során meg kell adnunk az asszociáció navigációs irányait. Az interakció-diagramokon minden üzenetküldés esetén szükséges, hogy a megfelelő kapcsolat navigálható legyen a küldőtől a fogadó objektum irányába.

Az attribútumok, műveletek és asszociációk segítségével már könnyen *azonosíthatjuk az* osztályok közötti *általánosítás-pontosítás viszonyokat*, illetve meghatározhatjuk a már felvett osztályok alapján azok általános változatait, amelyekbe a közös, ismétlődő jellegzetességeket kiemelhetjük. A jelölésnek a programozási nyelven elérhető konkrét technikára kell utalni, mely objektumorientált nyelv esetén általában az öröklés `(inheritance)`. Öröklés hiányában az általánosítás más módon, pl. delegációval is megvalósítható.

Az osztály műveleteinek felsorolása mellett a *művelet* megvalósítási módját is meg kell határoznunk. Közismert technika esetén a megvalósítást megadhatjuk egy rövid utalással (pl. bináris keresés), egyébként pszeudokódot vagy természetesnyelvű leírást alkalmazunk.

Bizonyos tervezési objektumok működését állapot-vezérelt módon adhatunk meg, ami azt jelenti, hogy adott üzenetekre az objektum az állapotától függően más és más fajta válaszokat adhat. Az ilyen működés leírására az egyik legjobb eszköz az állapot-diagram. A tervezés során meg kell adnunk, hogy az egyes állapotokat milyen attribútum-érték kombinációnak feleltetjük meg.

Az osztályok tervezése közben össze kell gyűjtenünk az osztállyal kapcsolatos *speciális* (pl. nem-funkcionális) *követelményeket*, melyeket figyelembe kell venni az osztály implementációjakor.

Alrendszer tervezése

Az alrendszer tervezésekor *ellenőriznünk* kell az *alrendszer függőségeit*, hogy az alrendszer a lehető legkevésbé függjön a többi részrendszertől.

A tervezés további feladata az *alrendszer interfészeinek ellenőrzése*. Az interfészek által definiált műveleteknek teljesítenie kell az összes olyan, az alrendszerre vonatkozó követelményt, melyet a különböző használati eset megvalósítások során játszott szerepe megkövetel.

A tervezés során az *alrendszerek tartalmát* is *ellenőrizni* kell, hogy az interfészekben meghatározott műveletek megvalósítása a követelményeknek megfelelően hajtódik-e végre.

Összefoglalás

A tervezésnek a készítendő rendszerről teljes mélységében részletes képet kell adnia, a kiválasztott programozási nyelvekre és technológiákra vonatkozóan is, így meg kell adnia az implementációhoz szükséges dokumentumokat, annak ütemezéséhez a rendszert kezelhető részekre kell bontania és meg kell határoznia a különböző részek közötti interfészeket. A tervezésnek meg kell rajzolni a rendszer implementációjának a vázát, így egy teljesen konkrét fizikai modellt kell meghatároznia, amely az implementáció „alaprajza” `(blueprint)` lesz.

A tervezés központi produktuma egy fizikai szintű osztálydiagramként megadott tervezési modell, amely a használati esetek fizikai szintű megvalósításait írja le. A tervezési modellt tervezési rendszerek és alrendszerek segítségével bontjuk fel kezelhető részekre. A tervezés másik eredménye a telepítési modell, amely a rendszer részeinek az ún. csomópontokon (számítógépeken vagy más eszközökön) való fizikai elhelyezkedését ábrázolja.

A tervezés első lépése az architekturális tervezés, mellyel az implementációs alapszerkezeteket határozzuk meg. Ezen keretek tartalmát a használati esetek tervezési szintű megvalósításával töltjük fel, majd az osztályokra fókuszálva összegyűjtjük az azokkal kapcsolatos implementációs követelményeket.

Kérdések

- Hogyan foglalhatjuk össze a tervezés célját?
- Mi a tervezési modell és milyen elemekből épül fel?
- Mi jellemző a tervezési osztályok megadására?
- Mi a tervezési rendszer/alrendszer?
- Mit tartalmaz a tervezési modell arhitekturális nézete.
- Mit tartalmaz a telepítési modell?
- Milyen lépésekben zajlik az arhitekturális tervezés?
- Hogyan végezhető el egy használati eset tervezési szintű megvalósítása?
- Hogyan azonosítjuk egy osztály műveleteit?
- Milyen módon kell ellenőriznünk a tervezési alrendszereket?

Implementáció

Az implementáció munkafolyamatában az elemzés, majd a tervezés során összeállított dokumentumokból kiindulva a rendszert ún. *komponensekként* implementáljuk, mely komponensek a Unified Process terminológiája szerint forráskódokat, szkripteket, bináris állományokat, futtatható programokat, dokumentumokat, képeket, stb. jelentenek.

Az implementáció céljait a következő pontokban foglalhatjuk össze:

- Az iterációk végén megkövetelt *rendszer-integrációk* tervezése.
- Az osztottság tervezése a rendszer futtatható programjainak csomópontokhoz történő rendelésével.
- A tervezési osztályok és alrendszerek implementálása, mely nagyobb részben a forráskódok előállítását jelenti.
- A komponensek önálló tesztje és integrációjuk egy vagy több futtatható programmá. Ezt a lépést a későbbi teszt során a rensdzer-integráció és az együttes teszt követi.

Implementációs modell

Az implementáció munkafolyamatában előállított implementációs modell a működő, azaz az implementációs rendszert, illetve annak implementációs részrendszereit jelenti, melyek komponenseket és interfészeket tartalmaznak.

A *komponens* `(component)` adott modellelemek fizikai egysége. A komponens típusát sztereotípiaként adhatjuk meg, pl.:

- `executable`: végrehajtható program
- `file`: forráskódot vagy adatot tartalmazó állomány
- `library`: statikus vagy dinamikus programkönyvtár
- `table`: adatbázistábla
- `document`: alapvetően szöveget tartalmazó dokumentum

Az implementáció során gyakran alkalmazunk ún. *csonkokat* `(stub)`, melyek valamely más komponens redukált funkcionalitású változatai. A csonkok segítségével egyszerűsíthetjük és áttekinthetőbbé tehetjük az új komponensek bevezetését. Csonkokat gyakran használunk teszt- illetve szimulációs célra.

Az implementációs modellnek is létezik *architekturális nézete*, mely az architektúra leírás része lesz. Az architekturális nézetben a következők találhatók meg:

- Az implementációs modell alrendszerekre történő felbontása, a közöttük lévő függőségek és az általuk megvalósított interfészek. Mivel az implementációs modellnek közvetlenül meg kell felelni a tervezési modell elemeivel, ezért az implementáció során nem szükséges az alrendszerekre vonatkozó információkat újra megadni.
- Kulcsfontosságú komponensek, melyek központi szerepűek a rendszer működése szempontjából.

Munkatársak

Az implementáció során az *architektúra tervező* `(architect)` a felelős az implementációs modell egészéért, annak helyes, konzisztens és áttekinthető formájáért. A modell helyes, ha teljesíti a korábbi munkafolyamatokban megfogalmazott követelményeket, beleértve a funkcionális és a nem-funkcionális követelményeket is, valamint az elemzés és tervezés során kidolgozott alapszerkezetet követi. Az implementáció lényeges lépése bizonyos (pl. futtatható) komponenseknek a telepítési modellben meghatározott csomópontokhoz történő rendelése, mely ugyancsak az architektúra tervező feladata.

A *komponens mérnök* `(component engineer)` dolgozza ki és kezeli egy vagy több komponens forráskódját, biztosítva, hogy az a megkövetelt funkcionalitást valósítja meg. A komponens mérnök feladata lehet egy vagy több implementációs alrendszer kezelése, integritásának biztosítása is.

A *rendszer integrátor* `(system integrator)` feladata a komponensekként implementált rendszer egészének összeállítása. Ez jelenti a részek összeépítési sorrendjének a meghatározását, valamint a részek integrálását, amint azok implementálása befejeződött.

Munkafolyamat

Architekturális implementáció

Az architekturális implementáció során meghatározzuk és kiemeljük az implementációs modell architektúra szempontjából lényeges elemeit. Ez a munkafolyamat alapvetően két lépésre bontható.

Az *architektúra szempontjából lényeges komponenseket azonosítjuk*, amelyek például a futtatható programok. Ezeket a komponenseket célszerű a lehető legkisebb számban kiválasztani, mindössze azokat, amelyek a működő alkalmazás központi szerepű részei lesznek.

Második lépésben a *komponenseket csomópontokhoz rendeljük*, a korábbi munkafolyamatokban meghatározott hálózati konfigurációnak megfelelően.

Rendszer integráció

A rendszer integráció során össze kell állítani egy integrációs tervet `(integration plan)`, mely meghatározza az iteráció során kifejlesztendő részeket, valamint minden egyes részre az azzal kapcsolatos követelményeket. A rendszerintegráció másik feladata a részeknek a teljes tesztet megelőző integrálása. A fejlesztés folyamán ez a tevékenység ezért két lépésre bontható.

A *soron következő rész tervezése* annak a meghatározását jelenti, hogy a következő részbe mely komponensek, illetve abba mely modellelemek kerüljenek bele. Az

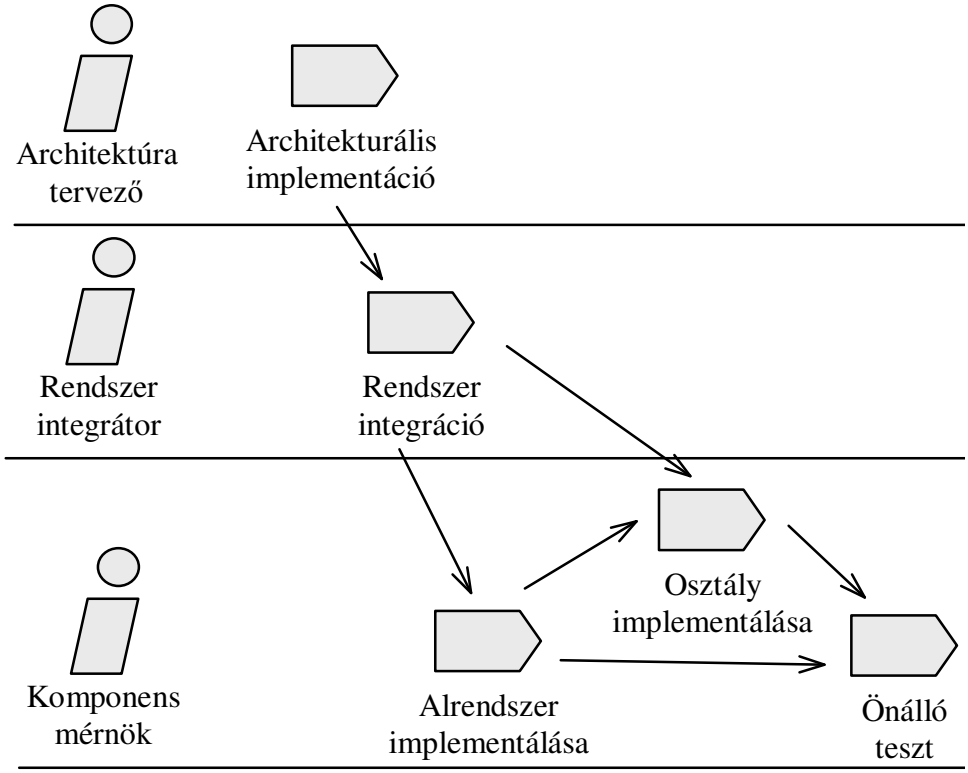
inkrementális fejlesztés miatt a „rész” általában nem egy újonnan és függetlenül kifejleszthető egységet jelent, hanem a rendszer egy újabb és bővebb belső változatát.

A rendszer integráció befejező lépése *a rész integrálása*, mely a későbbi együttes teszt kiindulópontja lesz.

Alrendszer implementálása

Az alrendszer implementálása során biztosítani kell, hogy az alrendszer alkalmas az integrációs tervben meghatározott célok megvalósítására, mely a használati esetekként megfogalmazott követelményeket már a tervezés munkafolyamatában meghatározott keretek között teljesíti.

Az implementáláskor célszerű újra *ellenőrizni az alrendszer tartalmát*, hogy a benne szereplő komponensek alkalmasak-e az alrendszer feladatának ellátására.



14. Ábra: Implementáció

Osztály implementálása

Az osztály implementálása egy kiválasztott tervezési osztály komponensként történő megvalósítását jelenti.

Először *körvonalazzuk a komponenst*, melyben megadjuk a tervezési osztály forráskódját. A kezelhetőség miatt gyakran egyetlen tervezési osztályt egyetlen komponensként adunk meg, bár több nyelv is lehetővé teszi, hogy egy állományban több osztályt is definiáljunk.

A következő tevékenység a *kód generálása a tervezési osztályból*, melyet azért nevezünk generálásnak, mivel a tervezés során specifikált osztály gyakorlatilag közvetlenül a programozási nyelv szintaktikájának megfelelő formává alakítható. Elvileg hasonló leképezés, illetve generálási módszer az asszociációk és aggregációk esetén is megadható, ez azonban a programozási nyelv eszközei mellett a kiválasztott technikáktól is nagymértékben függ.

Az osztály vázának elkészítése után *implementáljuk a műveleteket*. Az implementáció a megfelelő metódusok elkészítését jelenti, mely esetén ki kell választani a megfelelő algoritmust és adatszerkezetet, majd a programozási nyelven kódolni kell a műveleteket.

Az osztály implementálása során a *komponensnek teljesítenie kell a megfelelő interfészeket*, amelyeket a tervezési osztályban definiáltunk.

Önálló teszt végrehajtása

A tevékenység során az implementált komponenst önállóan, egységében teszteljük, hogy szolgáltatja-e a szükséges funkcionalitást. A tesztelésnél általában két módszert alkalmazunk.

A *specifikációs* („fekete-doboz”) *teszt* esetén a komponens működését a kívülről megfigyelhető viselkedése alapján teszteljük, figyelmen kívül hagyva azt, hogy a működés milyen módon lett implementálva. Ehhez a bemenet, kimenet és a komponens állapotának kombinációira ekvivalencia-osztályokat kell meghatároznunk, melyek tetszőleges eleme esetén a komponensnek hasonló módon kell viselkedni.

A strukturális („fehér-doboz”) teszt esetén a komponenst belsőleg, az implementáció alapján ellenőrizzük, mely a teljes forráskód ellenőrzését jelenti.

Összefoglalás

Az implementáció során a rendszert komponensekként implementáljuk. Ezen kívül meg kell terveznünk az iterációk végén megkövetelt rendszer-integrációkat, az osztottságot és elő kell állítanunk a tervezési osztályok és alrendszerek implementációját. Az implementáció jelenti még a komponensek önálló tesztjét és integrációjukat egy vagy több futtatható programmá.

Az implementációs modell alapelemei a komponensek, valamint annak is létezik egy összefoglaló jellegű architekturális nézete.

Kérdések

- Foglalja össze az implementáció céljait!
- Mondjon példákat a komponenseknél megadható sztereotípiákra!
- Mit tartalmaz az implementációs modell architekturális nézete?
- Hogyan történik egy osztály implementálása?
- Tesztelésnél alapvetően mely két módszert alkalmazzuk?

Teszt

A teszt munkafolyamat célja minden egyes implementált rész tesztelése, melyen a rendszer közbülső, belső változatait és a végleges, kibocsátott rendszert is értjük. A tesztelés alapvető céljai a következők:

- Minden egyes iteráció által igényelt teszt összeállítása, beleértve az integrációs és a rendszer teszteket.
- A tesztek tervezése és implementálása, mely során teszt-esetekben adjuk meg, hogy mit kell tesztelni, teszt-eljárásokban adjuk meg a tesztelési módokat és ha lehetséges, teszt-komponenseket készítünk a tesztek automatikus végrehajtására.
- A teszteket végre kell hajtani és azok eredményeit szisztematikusan ki kell értékelni. A felfedezett hibák és hiányosságok más munkafolyamatok tevékenységeinek ismételt végrehajtását igényelhetik.

Teszt-modell

A teszt-modell foglalja össze az implementációs modell során kidolgozott futtatható komponensek tesztelési módját, beleértve az integrációs és rendszer teszteket.

A teszt-modell a következő elemekből épül fel:

- A teszt-eset *(test case)* a tesztelés egy módját határozza meg, azt hogy *mit* kell tesztelni, és azt milyen bemenet, eredmények és feltételek mellett.
- A teszt-eljárás *(test procedure)* megadja egy vagy több test-eset, vagy azok bizonyos részeinek végrehajtási módját, a *„hogyant”*. A teszt-eljárás lehet a tesztelőnek megadott utasítás-sorozat, vagy automatikus teszt esetén a teszt-eszközzel végrehajtandó interakció-sorozat, mely alapján teszt-komponens készíthető.
- A teszt-komponens *(test component)* alkalmas egy vagy több teszt-eljárás vagy annak részeinek automatizált végrehajtására. A teszt-komponenst valamely szkript-nyelven adjuk meg, leprogramozzuk valamely programozási nyelven, vagy felvesszük egy alkalmas teszt-automatizálási eszközzel.

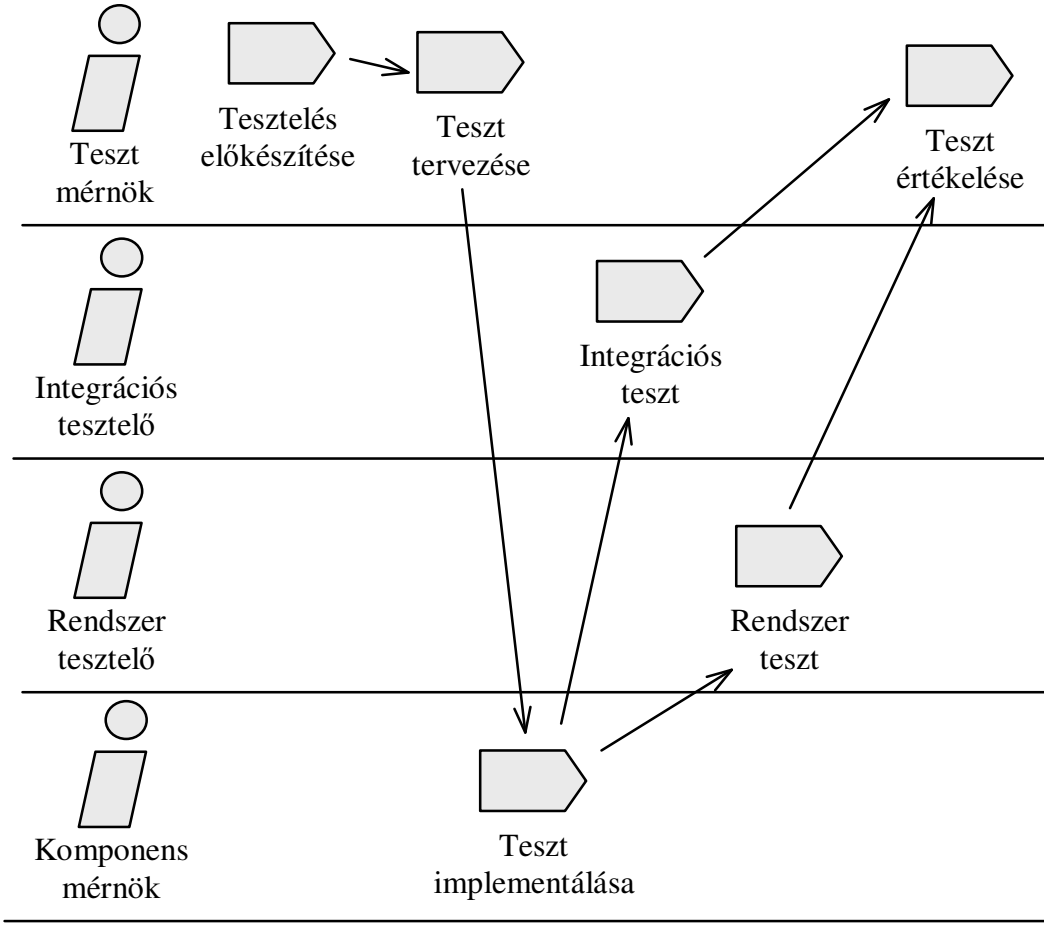
Munkatársak

A *teszt mérnök (test planner)* feladata a teszt-modell integritásának ellenőrzése, biztosítva, hogy az valóban alkalmas a rendszer hibáinak és hiányosságainak a kiszűrésére. A teszt mérnök összeállítja a teszteket, azok céljainak meghatározásával és végrehajtásuk ütemezésével. A teszt mérnök kiválasztja és leírja a teszt-eseteket és a szükséges teszt-eljárásokat, végül kiértékeli azok eredményét.

A *komponens mérnök (component engineer)* feladata az automatikus teszt-komponensek implementálása.

Az *integráció tesztelő* `(integration tester)` hajtja végre az integrált részek együttes tesztjét, azt vizsgálva, hogy az újonnan implementált elemek a rendszer egészében a megfelelő módon működnek.

A *rendszer tesztelő* `(system tester)` a rendszer egészének, kívülről megfigyelhető működésének az ellenőrzését végzi, így elsősorban az aktor és a rendszer közötti interakciókat vizsgálja. A tesztek a használati esetek ellenőrzésére alkalmas teszt-eseteket követik. Feladatkörénél fogva a rendszer tesztelőnek nem szükséges ismerni a rendszer belső felépítését.



15. Ábra: Teszt

Munkafolyamat

Tesztelés előkészítése

A tesztelés előkészítése a következő lépésekre bontható:

- Meghatározzuk a tesztelési stratégiát, pl. legalább a tesztek 80%-nak automatikusnak kell lenni, a többi lehet kézzel végrehajtott; minden használati eset normál lefolyását és három alternatív lefolyását tesztelni kell; a teszt eredményesnek tekinthető a teszt-esetek 90%-ának sikeressége esetén.

- Megbecsüljük a tesztelés erőforrás-igényét, emberi és rendszer-erőforrásonként is részletezve.
- Ütemezzük a teszt-tevékenységeket.

Teszt tervezése

Az előkészítést követően meg kell határozni és meg kell *tervezni a teszteket*, a tesztek jellegének megfelelő módon és lépésekben.

Az *integrációs teszt-esetek tervezésének* a célja a rendszerbe integrált új komponensek egymás közötti interakcióinak az ellenőrzése. A legtöbb integrációs teszt a használati esetek tervezési szintű megvalósításaiból következik, mivel azok az aktorok és objektumok közötti interakciókat írják le.

A *rendszer teszt-esetek tervezésével* a rendszer együttes működését ellenőrizzük különböző feltételek mellett, mely feltételek jelenthetnek különböző hardware környezeteket (processzor-sebesség, központi memória és merevlemez kapacitás), illetve software környezeteket (más adatbáziskezelő, más Web-böngésző, stb.).

A tesztek speciális változatai a *regressziós tesztek*, melyek eredetileg a rendszer korábbi változatainak tesztjei voltak, az aktuális iterációban pedig a szerepük az, hogy ellenőrizzék, hogy a korábban már kifejlesztett és ellenőrzött funkcionalitás nem sérül a módosítással és az újabb elemek beemelésével.

A teszt-esetek sorravételével azokra teszt-eljárásokat kell megadni. A teszt-eljárások kidolgozásakor célszerű a már korábban kidolgozott eljárásokat felhasználni.

Teszt implementálása

A teszt implementálása az automatizáltan végrehajtható teszt-komponensek implementációját jelenti. A teszt-komponenseket vagy egy teszt-automatizálási eszközzel készítjük el, vagy arra külön programot kell összeállítanunk. A teszt-komponensek általában nagymennyiségű bemeneti adattal dolgoznak és nagymennyiségű eredményadatot állítanak elő.

Integrációs teszt végrehajtása

Az integrációs tesztek során az újonnan kifejlesztett részeket és a rendszer egészének együttes működését vizsgáljuk.

Rendszer teszt végrehajtása

A rendszer teszt a rendszer egészét, kívülről megfigyelhető működését ellenőrzi, így elsősorban az aktor és a rendszer közötti interakciók vizsgálata a célja.

Teszt értékelése

A teszt értékelésénél alapvetően két szempontot kell figyelembe vennünk. A teszt *teljessége* a kód ellenőrzött részének százalékosan megfogalmazható arányát jelenti. A rendszer *megbízhatósága* a sikeres tesztek arányával fogalmazható meg.

Összefoglalás

A teszt célja az implementált részek tesztelése, melyen a rendszer közbülső, belső változatait és a végleges, kibocsátott rendszert is értjük. Mindez jelenti az egyes iterációk által igényelt tesztek összeállítását, a tesztek tervezését és implementálását, valamint a tesztek kiértékelését.

A teszt előkészítésénél meghatározzuk a tesztelési stratégiát, megbecsüljük annak erőforrásigényét és ütemezzük a teszteket. A teszteket megtervezzük és implementáljuk, majd végrehajtjuk az integrációs és a rendszer teszteket, végül kiértékeljük az eredményeket.

Kérdések

- Melyek a tesztelés alapvető céljai?
- Milyen elemeket tartalmaz a teszt-modell?
- Mit jelent a tesztelési stratégia meghatározása?
- Mit jelent a regressziós teszt?
- Mit jelent a rendszer megbízhatósága?