21 SZOFTVEREVOLÚCIÓ

TÉMAKÖRÖK

A fejezet célja a szoftverevolúció bemutatása, illetve a szoftvermódosítás néhány lehetséges útjának leírása. A fejezet elolvasása után:

- érteni fogjuk, hogy a változtatás elkerülhetetlen annak érdekében, hogy a szoftverrendszerek hasznosak maradhassanak, illetve hogy a szoftverfejlesztés és -evolúció egy spirális modellbe integrálható;
- megismerjük a szoftver karbantartásának különböző módjait, és megismerjük a szoftverkarbantartás költségeit befolyásoló tényezőket;
- tisztában leszünk a szoftverevolúcióba ágyazott folyamatokkal, beleértve a szoftver újratervezését;
- képesek leszünk megérteni az ősrendszerek esetében annak az eldöntését, hogy ezeket el kell-e dobni, karban kell-e tartani, újra kell-e fejleszteni, vagy le kell-e cserélni.

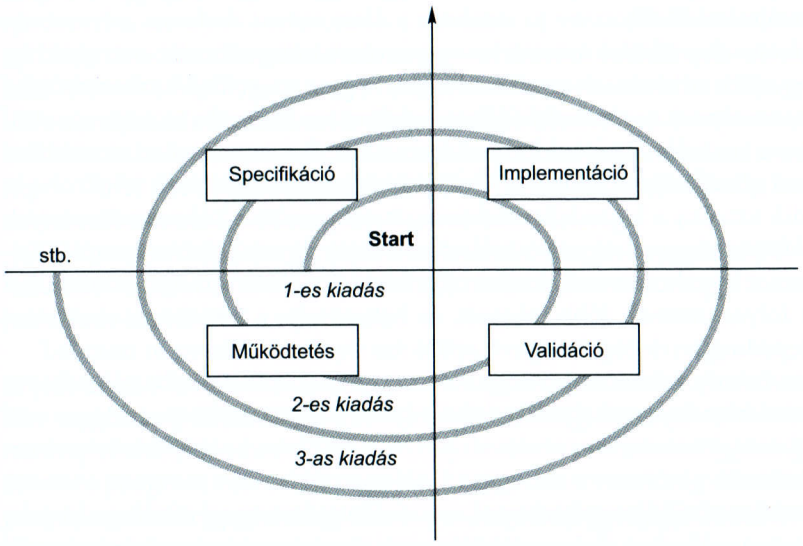
TARTALOM

A rendszerek telepítését követően elkerülhetetlenül változásokon kell átesniük a használhatóság megőrzésének érdekében. Amint egy szoftvert használni kezdenek, új követelmények fogalmazódnak meg vele szemben, a meglevő követelmények pedig változnak. Az üzleti változások gyakran gerjesztenek új követelményeket a meglevő szoftverekkel szemben. A szoftver bizonyos részeinél elképzelhetők változtatások az üzemeltetés során észrevett hibák kijavítására, esetleg azért, hogy egy új platform alá adaptálható legyen, vagy azért, hogy a teljesítményét növeljük, de ezeken kívül még további, nemfunkcionális jellemzők miatt is. A szoftverfejlesztés ezért a rendszer átadásával nem fejeződik be, hanem végigkíséri az egész életciklusát.

A szoftverevolúció fontos, miután a szervezetek mostanra teljesen a szoftverrendszerüktől függenek, és már több millió dollárt fektettek ezekbe a rendszerekbe. A szervezetek számára a rendszereik létfontosságú üzleti előnyt jelentenek, és ezen előnyök fenntartásáért további befektetéseket kell áldozniuk a rendszerek megváltoztatásába. A nagy cégek szoftverköltségvetésének nagy részét a létező rendszerek fenntartásának szentelik, és ezért nem kell meglepődnünk, hogy a költségek 90 százaléka evolúciós költség (Erlikh, 2000). Nagy a bizonytalanság ebben a százalékban, mivel az emberek különböző dolgokra gondolnak, amikor evolúciós vagy karbantartási költségekről beszélnek.

Ahogyan majd később tárgyalom, a telepítés utáni változtatások nem egyszerűen a szoftverhibák javítását érintik. A változtatások nagy része az új követelmények következménye, amelyeket az üzleti és felhasználói igények változása váltott ki. Következésképpen úgy gondolhatunk a szoftverfejlesztésre, mint egy spirális folyamatra, amely a követelmények felméréséből, a tervezésből, az implementációból és a tesztelésből áll, és ami végigköveti a rendszer életciklusát. Ezt láthatjuk a 21.1. ábrán. A rendszer 1. verziójának megalkotásával indítunk. Amint leszállításra került, változásokat indítványoznak, és a 2. verzió fejlesztése szinte rögtön elkezdődik. Gyakorlatilag az evolúció szükségessége nyilvánvalóvá válhat, mielőtt a rendszert telepítenék, így az új verziók fejlesztése megkezdődhet még mielőtt a kezdeti verziót kibocsátanák.

Ez a szoftverevolúció idealisztikus modellje, amely olyan szituációkban alkalmazható, amikor egyetlen szervezet felelős a kezdeti szoftver fejlesztéséért és a szoftver evolúciójáért is. Általánosságában ezt a megközelítést használják a szoftverfejlesztéshez. Előfordulhat azonban, hogy a megrendelt szoftvert külső cég fejleszti, de a szoftverevolúcióért a megrendelő saját fejlesztőcsapata a felelős. Esetleg a szoftver felhasználója külön megegyezést köthet egy külső céggel a rendszer támogatására és evolúciójára.



21.1. ábra. A fejlesztés és az evolúció spirális modellje

Ebben az esetben gyakran vannak megszakítások a spirális folyamat folytonosságában. Előfordulhat, hogy az egyik cég a követelményeket és a tervezési dokumentációt nem küldi el a másiknak. Cégek egyesülhetnek, újraszerveződhetnek vagy örökölhetnek szoftvereket más cégektől, és végül úgy találhatják, hogy változtatni kell. Amikor a szoftverfejlesztés és -evolúció között észrevehető az átmenet, a kibocsátás utáni szoftver változtatását gyakran *szoftverkarbantartás*nak nevezik. Ahogy a fejezetben még később tárgyalom, a karbantartás a normál szoftverfejlesztéshez képest további folyamatok tevékenységeit vonja maga után, mint például magának a programnak a megértése.

21.1. A PROGRAMEVOLÚCIÓ DINAMIKÁJA

A programevolúciós dinamika a rendszerváltozások tanulmányozása. Az ebben a tárgykörben készült munkák nagy részét Lehman és Belady adta ki, először az 1970-es és 1980-as években (Lehman és Belady, 1985). A munka az 1990-es években folytatódott, amikor Lehman másokkal együtt az evolúciós folyamatok visszajelzéseinek jelentőségét kutatta (Lehman, 1996; Lehman *és mások*, 1998; 2001). Ezekből a tanulmányokból állították össze a rendszer változtatásának szabályait (Lehman-törvények). Azt állítják, hogy ezek a törvények (igazából hipotézisek) invariánsak és széleskörűen alkalmazhatók. Lehman és Belady számos nagy szoftverrendszernél vizsgálták a növekedést és evolúciót. A 21.2. ábrán a mérésekből származó törvények láthatók.

Az első törvény állítása szerint a rendszerkarbantartás egy elkerülhetetlen folyamat. A rendszer környezetének változásával új követelmények lépnek fel, és a rendszert módosítani kell. A módosított rendszernek az adott környezetbe történő újrabevezetése további környezeti változásokat von maga után, így az evolúciós folyamat újrakezdődik.

A második törvény állítása szerint, ha egy rendszer megváltozik, a struktúrája romlik. Az egyetlen mód ennek az elkerülésére, hogy a megelőző karbantartásba invesztálunk, amelynek során kellő időt szánunk a szoftver struktúrájának a tökéletesítésére, a funkcionalitásának növelése nélkül. Értelemszerűen ez többletköltséget jelent a szükséges rendszerváltoztatások implementálásán felül.

A harmadik törvény a legérdekesebb és legtartalmasabb a Lehman-törvények között. Azt állítja, hogy a nagy rendszereknek saját dinamikája van, ami a fejlesztési folyamat egy korai szakaszában jön létre. Ez determinálja a rendszerkarbantartás folyamatainak főbb irányait, és behatárolja a lehetséges rendszerváltoztatások számát.

Lehman és Belady azt mondja, hogy a törvény azoknak a strukturális tényezőknek a következménye, amelyek befolyásolják és korlátozzák a rendszer változtatását ugyanúgy, mint a szervezési tényezők, amelyek az evolúciós folyamatra vannak hatással.

Amint a rendszer túllép egy bizonyos minimális méretet, egyre nehezebb lesz megváltoztatni. A nagysága és komplexitása miatt nehéz a rendszert megérteni,

Törvény	Leírás
Folyamatos változás	Egy valós környezetben használt programnak szükségszerűen meg kell változnia, különben fokozatosan veszít az adott környezetben történő használhatóságából.
Komplexitás növelése	Ahogy egy fejlesztés alatt álló program változik, a struktúrája egyre komplexebb lesz. Extra erőforrásokat kell a struktúra megőrzésének és egyszerűsítésének szentelni.
Nagy programok fejlődése	A programevolúció egy önszabályzó folyamat. A rendszer tulajdonságai, mint a méret, a kiadások közötti idő és a jelzett hibák nagyjából invariánsak az egyes verziókra.
Szervezeti stabilitás	Egy program élettartama alatt a fejlesztés rátája nagyjából konstans, és független a rendszer fejlesztésére fordított erőforrásoktól.
Jártasság megőrzése	A rendszer élettartama alatt a járulékos változások az egyes verziókban nagyjából azonosak.
Folytatólagos növekedés	A rendszerek által nyújtott funkcionalitásnak folyamatosan nőnie kell a felhasználói megelégedés fenntartása végett.
Minőség csökkenése	A rendszerek minősége csökkenni látszik mindaddig, míg hozzá nem illesztik a működési környezethez.
Visszacsatolási rendszer	Az evolúciós folyamatok magukban foglalnak többtényezős, többciklusos visszacsatolási rendszereket, amelyeket visszacsatolási rendszerként kell kezelnünk, hogy jelentős termékjavulást érjünk el.

21.2. ábra. Lehman-törvények

a programozók sokkal könnyebben hibázhatnak, ami rendszerhibák megjelenését okozhatja. Ezért, ha csak kicsiket változtatunk a rendszeren, elkerülhetjük a megbízhatóság romlását. Egy nagy változtatás valószínűleg sok új hibát hozna a rendszerbe, amelyek korlátozzák a rendszer új verziójának hasznos újításait.

Nagy rendszereket általában nagy szervezetek készítenek, amelyeknek saját belső bürokráciájuk van, ami minden rendszer esetében beállítja a változtatási költségvetést és ellenőrzi a döntéshozatali folyamatot. A szervezeteknek döntést kell hozniuk a változtatások értékeiről és kockázatairól, illetve az ezzel járó költségekről. Az ilyen döntésekhez idő kell. Ez alatt az idő alatt pedig további, fontosabb rendszer-változtatási igények érkezhetnek.

Elképzelhető így, hogy az eredeti változtatásokat egy későbbi időpontig el kell halasztani. A szervezet döntéshozó folyamata ezért irányítja a rendszer változtatásának a fokát.

Lehman negyedik törvénye azt állítja, hogy a legtöbb nagyobb programozási projekt, egy általa *telített*nek nevezett állapotban működik. Azaz az erőforrásokban vagy a csapatban történő változtatás csak alig észrevehető hatást gyakorol a rendszer hosszú távú evolúciójára. Ez konzisztens a harmadik törvénnyel, amely szerint a program evolúciója nagyjából független a vezetőség döntéseitől. Ez a törvény megerősíti, hogy a nagy szoftverfejlesztő csapatok gyakran alacsony termelékenységűek, mert kommunikációs többletmunkák terhelik a csapat munkáját.

Lehman ötödik törvénye az egyes rendszerverzióknak a változásnövekedéseivel kapcsolatos. A rendszerhez új funkcionalitások hozzáadásával óhatatlanul új rendszerhibák jelennek meg. Minél több funkcionalitást adunk a rendszerhez egy verzióban, annál több hiba keletkezhet. Emiatt egy verzió nagy funkcionalitásbeli növekedése maga után vonja egy újabb verzió kiadását, amelyben az új hibák kerülnek kijavításra. Ebben a verzióban viszont relatíve kevés új funkcionalitás fog megjelenni. A törvény állítása szerint a nagy funkcionalitásbeli fejlesztéseknél figyelembe kell venni ezen hibák kijavításának a szükségességét.

Az első öt törvény Lehman kezdeti javaslataiból fogalmazódott meg, a többi a későbbi munkái során került be. A hatodik és hetedik törvény hasonló, és alapvetően azt állítják, hogy a szoftver felhasználói egyre inkább elégedetlenek lesznek a szoftverrel, hacsak karban nem tartjuk, illetve új funkcionalitást nem adunk hozzá. Az utolsó törvény a visszacsatolási folyamatokon végzett legújabb munkákat tükrözi, jóllehet az még nem világos, hogy ezt miként lehet hasznosítani a gyakorlati szoftverfejlesztésben.

Lehman észrevételei alapvetően ésszerűnek tűnnek. A karbantartási folyamat tervezésekor figyelembe kell venni ezeket. Megeshet, hogy az üzleti tényezők miatt ezeket alkalmasint figyelmen kívül kell hagyni. Például értékesítési okokból szükséges lehet, hogy több fontosabb rendszerváltoztatást végre kell hajtani egyetlen verzióban. Ennek lehetséges következménye, hogy egy vagy több verziót is rá kell szánni a hibajavításra.

Úgy tűnhet, hogy az alapvető különbségek a szoftvertermékek egyes kiadásai között megszegik Lehman törvényeit. Például a Microsoft Word egy 256K memóriával működő egyszerű szövegszerkesztőből egy gigantikus, újdonságokkal telerakott rendszerré vált. Mára rengeteg megabájt memóriát használ, és gyors processzort igényel a működéshez. Az evolúciója ellentmondani látszik Lehman negyedik és ötödik törvényének. Bár nekem az a gyanúm, hogy ez nem egy egyszerű program javított kiadásainak sorozata, sokkal inkább megtartották a nevet értékesítési szempontok miatt, de magát a programot az eredeti kiadás óta többször átírták és újrastrukturálták.

21.2. SZOFTVERKARBANTARTÁS

A szoftverkarbantartás a rendszer kiadását követően a rendszer megváltoztatásának egy alapvető folyamata. A kifejezést általában a szokványos szoftverekre alkalmazzák, ahol különálló fejlesztőcsapatok érintettek a kiadás előtt és a kiadás után.

A szoftveren végzett módosítások lehetnek egyszerű változtatások a kódolási hibák kijavítására, átfogóbb változtatások a tervezési hibák kijavítására vagy jelentősebb fejlesztések a specifikációs hibák kijavítására, esetleg új követelmények beillesztésére. A változásokat a létező rendszer komponenseinek módosításával, illetve ahol szükséges, új komponensek hozzáadásával lehet implementálni.

A szoftverkarbantartásnak három típusa van:

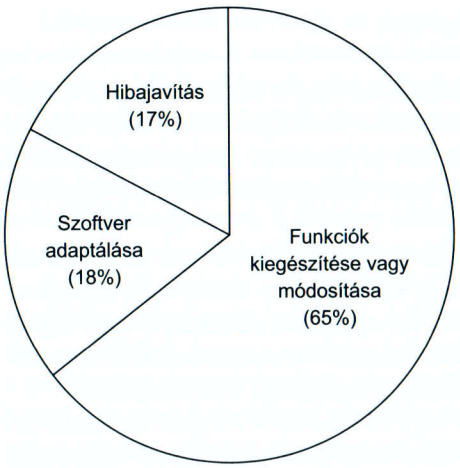
1. *A szoftverhibák kijavítására irányuló karbantartás.* Míg a kódolási hibák általában olcsón javíthatók, addig a tervezési hibák költségesebben, mert maguk után vonhatják más programkomponensek javítását is. A legköltségesebb a követelményekben lévő hibák javítása, mert szükségessé válhat a rendszer újratervezése is.
2. *A szoftver más működési környezethez történő adaptálására irányuló karbantartás.* Erre a karbantartásra akkor van szükség, ha bizonyos környezeti rendszertényezők, mint például a hardver, a platform operációs rendszere, vagy egyéb szoftverek változnak meg. Az alkalmazásrendszert annak érdekében kell módosítani, hogy együttműködhessen ezekkel a környezeti változásokkal.
3. *A rendszer funkcionalitásának kibővítésére vagy módosítására irányuló karbantartás.* Ez a karbantartás akkor szükséges, amikor szervezeti vagy üzleti változások hatásaként a rendszerrel szembeni követelmények megváltoznak. A szoftverrel szemben elvárt változtatások skálája általában sokkal nagyobb, mint a karbantartás többi típusánál.

A gyakorlatban a fenti karbantartási típusok között nincs éles határvonal. Amikor a rendszert egy új környezethez igazítják, előfordulhat új funkcionalitások felvétele annak érdekében, hogy kihasználják az új környezet tulajdonságait. Szoftverhibák gyakran azért fordulnak elő, mert a felhasználók előre nem látott módon használják a rendszert. A legjobb mód az ilyen hibák elkerülésére, ha az ő munkamódszerükhöz illesztjük hozzá a rendszert.

A karbantartás ezen típusai általában felismerhetők, de különböző emberek más és más nevet adnak ezeknek. *Korrektív karbantartás* alatt általában a hibák kijavítására irányuló karbantartást értjük. Az *adaptív karbantartás* jelentheti a szoftver más környezethez adaptálását, de jelentheti a szoftver új követelményekhez adaptálását is. A *perfektív karbantartás* jelentheti a szoftver tökéletesítését az új követelmények implementálásával, más esetekben a rendszer funkcionalitásának megtartását jelenti, tökéletesítve a struktúráját és teljesítményét. A névadási bizonytalanságok miatt elkerültem ezen kifejezések használatát az egész fejezetben.

Lientz és Swanson (Lientz és Swanson, 1980), illetve Nosek és Palvia (Nosek és Palvia, 1990) megfigyelései szerint a karbantartások kb. 65 százaléka az új követelmények implementálásával kapcsolatos, 18 százaléka az új operációs környezethez való adaptálással és 17 százalék a rendszer hibajavításával (21.3. ábra). Szokványos rendszerekre a költségek ilyen eloszlása nagyjából helytálló. A súlypont igazából nem a százalékos eloszláson van, hanem azon, hogy a rendszer hibajavítása nem a legköltségesebb folyamat. A rendszer új környezettel történő együttműködésének kifejlesztése, illetve az új vagy a megváltozott követelmények implementálása követeli meg a legnagyobb erőfeszítést.

A fejlesztési és karbantartási költségek közti arány alkalmazási területenként más és más. Guimaraes (Guimaraes, 1983) azt állítja, hogy az üzleti alkalmazásrendszerek karbantartási költségei hozzávetőlegesen összehasonlíthatók a rendszerfejlesztési költségekkel.



21.3. ábra. Karbantartási munkák eloszlása

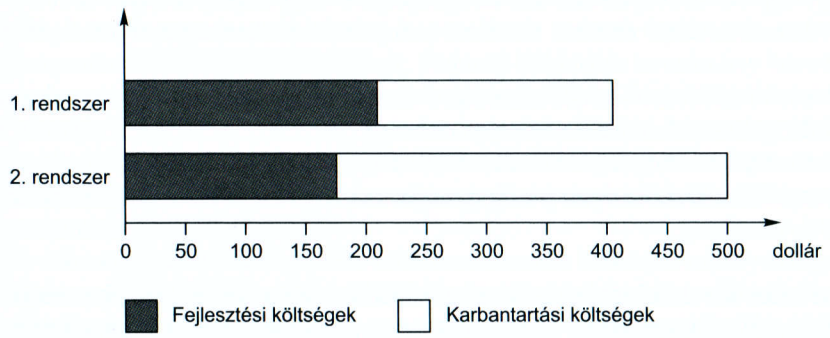
Beágyazott valós idejű rendszereknél a karbantartási költségek a fejlesztési költségek négyszeresét is kitehetik. A rendszer megbízhatóságával és teljesítményével szembeni magas elvárások gyakran azt jelentik, hogy a moduloknak szorosan kapcsolódniuk kell, és emiatt nehéz lehet megváltoztatni.

Általában költséghatékony, ha energiát áldozunk a rendszer tervezésébe és implementálásába a karbantartási költségek csökkentése érdekében. A kibocsátás után új funkcionalitás hozzáadása költséges, mivel időt kell fektetnünk a rendszer megértésébe, és elemezni kell a javasolt változtatások hatását. Ezért a fejlesztésbe fektetett munkával, ami könnyebben érthetővé és egyszerűbben változtathatóvá teszi a szoftvert, valószínűleg csökkenthetők a karbantartási költségek. A jó szoftvertervezési technikák, mint például a precíz specifikáció, az objektumorientált fejlesztés használata és a konfigurációkezelés, mind-mind hozzájárulnak a karbantartási költségek csökkentéséhez.

A 21.4. ábra azt szemlélteti, hogy a teljes élettartamköltségek csökkenhetnek, minél több energiát szánunk a fejlesztés alatt arra, hogy karbantartható rendszert készítsünk. A megértésben, elemzésben és tesztelésben rejlő potenciális költségcsökkenés miatt egy jelentős szorzót jelent, ha a rendszert karbantarthatóra tervezték. Az 1-es számú rendszernél 25 000 dollár extra költséget fektettek bele abba, hogy a rendszer jól karbantartható legyen. Ez a rendszer élettartalma alatt 100 000 dolláros megtakarítást eredményez a karbantartási költségekben. Feltételezve, hogy a fejlesztési költségek százalékos növelése arányos százalékban csökkenést okoz a rendszer teljes költségeiben.

Az egyik fontos ok, amiért a karbantartási költségek magasak, hogy új funkcionalitás hozzáadása egy működő rendszerhez sokkal költségesebb, mint ugyanazt a funkcionalitást implementálni a fejlesztés alatt. A kulcstényezők, amelyek elválasztják a karbantartást és a fejlesztést, és amelyek magasabb karbantartási költségekhez vezetnek, a következők:

1. *Csapat stabilitása.* A rendszer kibocsátása után általános, hogy felbomlik a tervezőcsapat, és tagjai új projekteken dolgoznak. A karbantartásért felelős új csapat vagy az egyének nem ismerik a rendszert, illetve a rendszertervezés során hozott döntések hátterét. Sok energiát vesz el a karbantartási folyamat során a létező rendszer megértése, még mielőtt a változtatásokat implementálni lehetne.
2. *Szerződéses felelősség.* A rendszer karbantartására általában külön szerződést kötnek, ami különbözik a fejlesztési szerződéstől. Lehet, hogy a karbantartási szerződést inkább más céggel kötik meg, nem az eredeti rendszerfejlesztővel. Ez a tényező a csapat stabilitásának hiányával együtt azt jelenti, hogy a fejlesztőcsapat számára nincs ösztönző, hogy olyan szoftvert írjanak, amelyet könnyű megváltoztatni. Ha a fejlesztőcsapat egy probléma kikerülésével energiát tud megspórolni, akkor ez számukra még akkor is jobban megéri, ha ez a karbantartási költségek növekedéséhez vezet.
3. *Személyi szakértelem.* A karbantartó csapat az alkalmazási területen gyakran tapasztalatlan és ismeretlen. A karbantartás így nem túl népszerű a szoftverfejlesztők körében. Úgy tekintenek rá, mint a szoftverfejlesztésnél kevesebb szakképzettséget igénylő munkára, és gyakran a legfiatalabb csapatra osztják ki. Ráadásul lehet, hogy a régi rendszereket elavult programozási nyelven írták, így a karbantartó csapatnál elképzelhető az adott nyelvet illető tapasztalatlanság, esetleg meg kell tanulniuk ezeket a nyelveket a rendszer karbantartásához.
4. *A program kora és struktúrája.* Ahogy a programok öregszenek, a változtatások miatt romlik a struktúrájuk, így nehezebb őket megérteni és módosítani. Néhány rendszert a modern szoftvertervezési technikák nélkül fejlesztettek. Talán ezek soha nem voltak jól strukturálva, és lehet, hogy inkább a hatékonyságra optimalizálták, mintsem az érthetőségre. Lehet, hogy elveszett a rendszer dokumentációja vagy inkonzisztens. Régi rendszerek esetében esetleg nem tervezték meg a konfigurációkezelést, így gyakran időpocsékolás megtalálni a megfelelő változtatandó rendszerkomponenst.



21.4. ábra. Fejlesztési és karbantartási költségek

Az első három probléma abból a tényből ered, hogy sok szervezet még mindig különálló tevékenységeknek tekintik a fejlesztést és a karbantartást. A karbantartás másodrendű tevékenység, és a fejlesztés alatt nincs ösztönzés arra, hogy pénzt fektessenek a későbbi változtatási költségek csökkentésére. Az egyetlen hosszú távú megoldás erre a problémára az, hogy elfogadjuk, hogy a rendszereknek ritkán van definiált élettartamuk, de meghatározhatatlan ideig folyamatosan használatban vannak valamilyen formában. Ahogy ajánlottam az előszóban, a rendszerekre úgy kell gondolni, hogy az élettartamuk alatt folyamatos fejlesztőmunka eredményeként folytonosan fejlődnek.

Bizonyos szempontokból a negyedik pontban említett leromlott struktúrájú rendszerek problémáját a legegyszerűbb megválaszolni. A szoftver-újratervezési technikák (a fejezet későbbi részében még röviden szólunk) alkalmazhatók a rendszer struktúrájának és érthetőségének a tökéletesítésére. Architekturális transzformációk adaptálhatják a rendszert új hardverhez. Megelőző karbantartási munkálatok (alapvetően inkrementális újratervezés) alkalmazhatók a rendszer tökéletesítésére és könnyebben változtathatóvá tételére.

21.2.1. Karbantartás előrejelzése

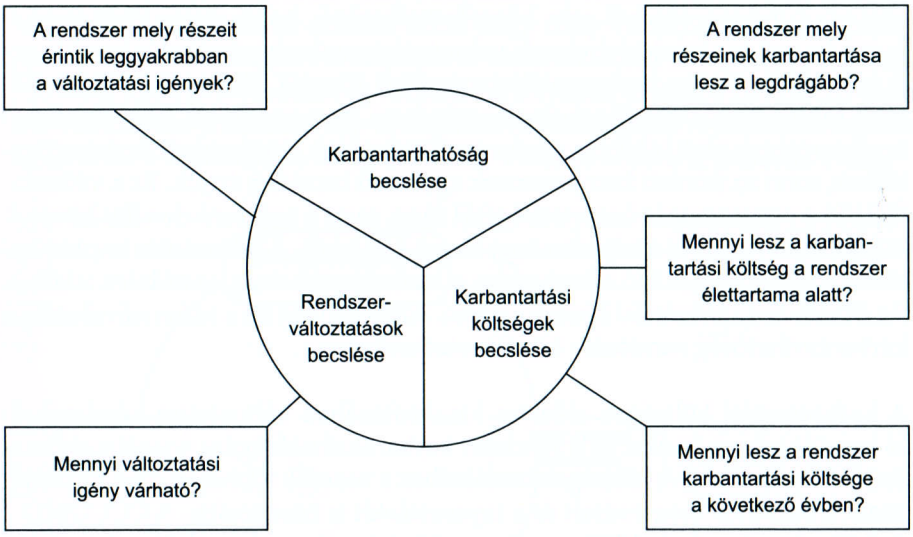
A vezetők nem szeretik a meglepetéseket, különösen ha azok nem várt nagyságú költségeket eredményeznek. Ezért érdemes megpróbálniuk előre jelezni a rendszer egy adott időszakra számított általános karbantartási költségeit, azt, hogy valószínűleg milyen rendszerváltoztatásokra lesz igény, és hogy várhatóan a rendszer mely részei okozzák a karbantartó személyzet számára a legsúlyosabb nehézségeket. A 21.5. ábra a különféle előrejelzéseket, valamint az azokkal kapcsolatos kérdéseket szemlélteti.

Ezek a különböző előrejelzések nyilvánvalóan szorosan kapcsolódnak egymáshoz:

1. Az, hogy egy rendszerváltoztatást elfogadjunk-e vagy sem, bizonyos mértékben a változtatás által érintett rendszerkomponensek karbantarthatóságától függ.
2. A rendszerváltoztatások implementálása általában rontja a rendszerstruktúrát, ennélfogva csökkenti a karbantarthatóságát.
3. A karbantartási költségek a változtatások számától függnek, a változtatások implementálásának költségeit pedig a rendszerkomponensek karbantarthatósága határozza meg.

A változtatási kérelmek számának megjóslásához meg kell értenünk a rendszer és annak külső környezete közötti kapcsolatot. Egyes rendszerek nagyon összetett kapcsolatban állnak a külső környezetükkel, ezért a környezet megváltoztatása elkerülhetetlenül változásokat eredményez a rendszerben. A rendszer és a környezete közötti kapcsolat megítéléséhez figyelembe kell venni a következőket:

1. *A rendszerinterfészek száma és komplexitása.* Minél nagyobb az interfészek száma és minél összetettebbek ezek az interfészek, annál valószínűbb, hogy igény lesz a változtatásra.
2. *Az öröklötten változékony rendszerkövetelmények száma.* Amint azt a 7. fejezetben tárgyaltam, valószínű, hogy a szervezeti politikákat és eljárásokat tükröző követelmények változékonyabbak, mint a stabil szakterületi jellegzetességeken alapuló követelmények.
3. *A vállalati folyamatok, amelyekben a rendszert használják.* A vállalati folyamatok fejlődése közben rendszer-változtatási igények lépnek fel. Minél több vállalati folyamatnál használjuk a rendszert, annál több változtatás válik szükségessé.



21.5. ábra. Karbantartás előrejelzése

A rendszer karbantarthatóságának becsléséhez meg kell értenünk a rendszer különböző komponensei közötti kapcsolatok számát és típusát, valamint eme komponensek belső komplexitását. Számos különféle tanulmány készült a rendszerben lévő komplexitás különböző típusairól (McCabe, 1976; Halstead, 1977), valamint a karbantarthatóság és a komplexitás közötti kapcsolatról (Kafura és Reddy, 1987; Banker *és mások,* 1993). A tanulmányok eredménye, mely szerint minél összetettebb egy rendszer vagy egy komponens, annál költségesebb a karbantartása, nem meglepő.

A komplexitási mérések fölöttébb hasznosnak bizonyultak olyan egyedi programkomponensek azonosításánál, amelyek karbantartása valószínűleg rendkívül költséges. Kafura és Reddy tanulmányukban (Kafura és Reddy, 1987) számos rendszerkomponenst vizsgáltak meg, és megállapították, hogy a karbantartási munkák általában néhány komplex komponensre koncentrálódtak. Ebből arra lehet következtetni, hogy anyagi szempontból érdemes lehet a különösen összetett rendszerkomponenseket egyszerűbb megoldásokkal helyettesíteni.

A rendszer üzembe helyezése után lehetőség nyílik a folyamat adatainak felhasználására, amelyek segítségével könnyebben becsülhetjük a karbantarthatóságot. A következőkben példákat láthatunk olyan folyamatmetrikákra, amelyek hasznosak lehetnek a karbantarthatóság becslésénél:

1. *A korrektív karbantartásra irányuló kérések száma.* A hibabejelentések számának emelkedése arra utalhat, hogy több hiba kerül a programba, mint amennyit a karbantartási folyamat alatt kijavítanak. Ez a karbantarthatóság csökkenését jelezheti.
2. *A hatásanalízishez szükséges átlagidő.* Ez azoknak a programkomponenseknek a számát tükrözi, amelyekre a változtatási kérelmek hatást gyakorolnak. Ezen időtartam növekedéséből arra következtethetünk, hogy egyre több és több komponenst érintenek a kérelmek, valamint hogy a karbantarthatóság csökken.
3. *A változtatási kérelem implementálására fordított átlagidő.* Bár ez és a hatásanalízisre fordított idő összefügghetnek egymással, nem azonosak. A szóban forgó tevékenységek alatt inkább a rendszerváltoztatások létrehozását és dokumentálását, mint az érintett komponensek egyszerű becslését értjük. Ez a változtatási idő a programozás összetettségétől függ, mert a nemfunkcionális követelményeknek (például a teljesítmény) eleget kell tenni. A változtatás implementálásához szükséges idő növekedése a karbantarthatóság romlására utalhat.
4. *Az elintézetlen változtatási kérelmek száma.* Amennyiben ez a szám növekszik, a karbantarthatóság romlására következtethetünk.

A karbantartási költségek előzetes kiszámításához változtatási kérelmekről szóló becsült információkat és a rendszer karbantarthatóságára vonatkozó becsléseket használunk fel. A költségek becsléséhez a vezetők többsége ezen információ mellett ösztönös megérzéseit és a tapasztalatát is hasznosítja. A COCOMO 2 modell (Boehm *és mások*, 1995), amelyet a 26. fejezetben tárgyalok, azt mutatja, hogy a szoftver-karbantartási munkák felbecsléséhez a már létező kód megértésére, valamint az új kód fejlesztésére fordított munkák alapul szolgálhatnak.

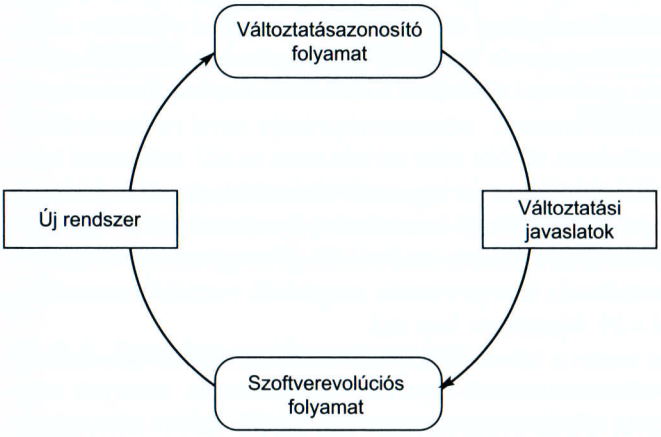
21.3. EVOLÚCIÓS FOLYAMATOK

A szoftverevolúciós folyamatok számottevő eltérést mutatnak annak függvényében, hogy milyen a karbantartandó szoftver típusa, a szervezet által használt fejlesztési folyamat, illetve a folyamatban részt vevő személyek. Néhány szervezetnél az evolúciós folyamat lehet informális folyamat, ahol a változtatási kérelmek a rendszerfejlesztők és a rendszer használói között folytatott beszélgetésekből adódnak. Más cégek esetében ez formális folyamat, a folyamat minden fázisában strukturált dokumentáció készítésével.

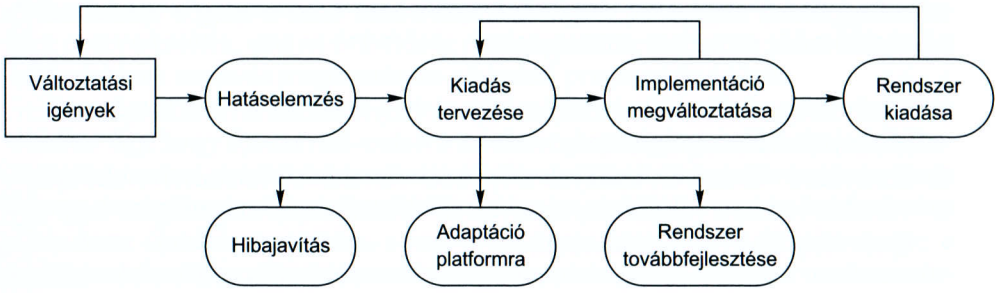
Minden szervezetnél a rendszer-evolúció vezérlőelemei a rendszer-változtatási javaslatok. Ezek a változtatási javaslatok magukban foglalhatnak már létező, de a kiadott rendszerben nem implementált követelményeket, lehetnek új

követelmények, esetleg a rendszerfenntartók kérelme a hibák javítására vagy a rendszer tökéletesítésére vonatkozó ötletek és javaslatok a rendszerfejlesztőktől. Ahogy a 21.6. ábra mutatja, a változások azonosításának és a rendszer-evolúciónak a folyamatai ciklikusak, és kihatnak a rendszer egész élettartamára.

Az evolúciós folyamat magában foglalja a változtatáselemzés, a kiadástervezés, a rendszer-implementáció és a vásárlóknak történő rendszer kiadásának alapvető folyamatait. A változtatások költsége és hatása megbecsülhető annak alapján, hogy a változtatás mennyire érinti a rendszert, illetve hogy mennyibe kerülhet a változtatás implementálása. Ha a javasolt változtatásokat elfogadják, elkezdődhet a rendszer új verziójának tervezése. A verzió tervezése alatt minden javasolt változtatást (hibajavítás, adaptáció, új funkcionalitás) figyelembe kell venni. Ezt követően kell dönteni arról, hogy mely változtatásokat implementálják a rendszer következő verziójában. A változtatásokat implementálják, validálják, majd kiadják a rendszer új verzióját. A folyamat ezután a következő verzióra tett változtatási javaslatok egy új halmazával ismétlődik. A 21.7. ábra áttekintést nyújt erről a folyamatról, amelyet Arthur írásából vettem át (Arthur, 1988).



21.6. ábra. Változtatásazonosítás és az evolúciós folyamatok

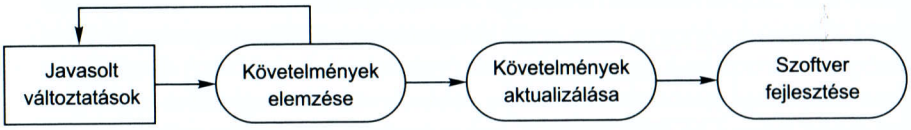


21.7. ábra. A rendszer-evolúció folyamata

A változtatás implementációjának a folyamata alapvetően a fejlesztési folyamat ismétlése, ahol megtervezik, implementálják és tesztelik a rendszerrel szemben végrehajtandó módosításokat.

Azonban itt alapvető különbség, hogy a változtatás implementációjának kezdeti lépése a program megértése. Ebben a fázisban meg kell érteni, hogy milyen a program struktúrája, illetve hogy miként látja el a funkcionalitását. A változtatás implementációja során ezt a tudást használjuk fel annak biztosítására, hogy az implementált változtatás a létező rendszert nem befolyásolja hátrányosan.

Ideális esetben a folyamatnak a változtatás implementációs lépése módosítja a rendszer specifikációját, a rendszertervet és a rendszer implementációját annak érdekében, hogy a rendszerváltoztatásokat tükrözze (21.8. ábra). A rendszerváltozásoknak megfelelő új követelményeket megtervezik, elemzik és validálják. A rendszerkomponenseket újratervezik és implementálják, illetve a rendszert újratesztelik. Ha ez megfelelő, a változtatáselemzési folyamat részeként a javasolt változtatások prototipizálása kivitelezhető.



21.8. ábra. Változtatás implementálása

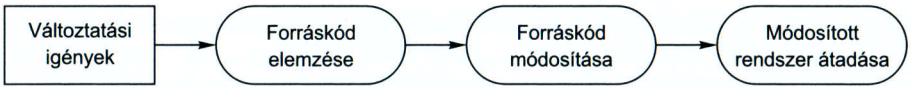
A szoftver változtatásával a rendszer egymást követő kiadásait fejlesztjük. Ezek a rendszerkomponensek verzióinak összetételeként alakulnak ki. Ezeket a verziókat nyomon kell követnünk annak érdekében, hogy megbizonyodhassunk, hogy minden rendszerverzióban a komponensek megfelelő verzióját használjuk. A konfigurációkezelésről a 29. fejezetben lesz szó.

Az evolúciós folyamat során a követelményeket részletesen kell elemezni, és gyakran felléphetnek a változtatásoknak olyan következményei, amelyek a korábbi változtatások elemzési folyamata során nem tűntek elő. Ez azt jelenti, hogy a javasolt változások módosulhatnak, illetve további, az ügyféllel történő egyeztetések szükségesek, még mielőtt implementálnánk ezeket.

A változtatási kérelmek alkalmanként olyan rendszerproblémákkal állnak összefüggésben, amelyeket sürgősen kezelni kell. Ezek a sürgős változtatások három ok miatt válhatnak szükségessé:

1. olyan rendszerhiba keletkezik, amelynek helyrehozása elkerülhetetlen a normális működés folytatásához;
2. környezeti változások, amelyek előre nem várt módon hatnak a rendszerre;
3. váratlan vállalati változások, amelyek új versenytársak megjelenése vagy egy új törvény miatt következhetnek be.

Ezekben az esetekben általában sokkal fontosabb a változtatás gyors elvégzése, mint a formális változtatási folyamat követésének biztosítása. A követel-



21.9. ábra. A sürgősségi javítás folyamata

mények módosítása és a tervezés helyett inkább a változtatásokat implementáló sürgősségi javítást készítenek a rendszer kódjához (21.9. ábra). Azonban ennek a szemléletnek az a veszélye, hogy a követelmények, a szoftver tervezése, valamint a kód fokozatosan összeegyeztethetetlenné válik. Ezt nehéz elkerülni, hiszen a változtatás gyors implementálása nehézségekbe ütközhet. Előfordulhat, hogy a karbantartó mérnököknek a szoftverhez új sürgősségi javítást kell készíteniük, és így a szabályos javítást elhalasztják. Ha a mérnök, aki megváltoztatta a kódot, elhagyja a csapatot még mielőtt a tervet aktualizálnák, a helyettese nehezen tudja újra összeegyeztetni a változtatásokat a követelményekkel és a tervvel.

További probléma a sürgősségi javításokkal, hogy a lehető leggyorsabban el kell azokat végezni. Ezért a rendszerstruktúra vonatkozásában a legjobb megoldás helyett egy megvalósítható megoldást is választhatunk. Ez felgyorsítja a szoftver öregedési folyamatát, ezért a jövőbeli változtatások fokozatosan nehezebbé válnak, a karbantartási költségek pedig emelkednek.

Ideális esetben a sürgősségi javítások elvégzésekor a változtatási kérelemnek elintézetlenül kellene maradnia a kódhibák kijavítása után. További elemzés után körültekintően lehet újraimplementálni. Természetesen a javítás kódját újra fel lehet használni. Ha az elemzéshez több idő áll rendelkezésre, előfordulhat, hogy a problémához egy másik, jobb megoldást találunk. Jóllehet ezek a változtatások a gyakorlatban alacsony prioritással fognak rendelkezni, mégis valószínűtlen, hogy ezeket visszavonják (különösen további változtatások esetén).

21.3.1. Rendszer újratervezése

Ahogy azt az előző alfejezetben már megbeszéltük, a rendszer-evolúció folyamata magában foglalja a változtatni kívánt program megértését, majd ezt követően a változtatások implementálását. Habár sok rendszert, különösen a régebbi (a 2. fejezetben ismertetett) ősrendszereket nehéz megérteni és megváltoztatni. Elképzelhető, hogy a programokat eredetileg a teljesítményre vagy a tárhely használatára optimalizálták, ami az érthetőség rovására ment, esetleg az idő múlásával a változtatások sorozata megbonthatta a kezdeti programstruktúrát.

Egy cég az ősrendszereinek változtatásával járó problémák leegyszerűsítésére dönthet úgy, hogy újratervezi ezeket a rendszereket, hogy a struktúrájuk és érthetőségük javuljon. Az ősrendszerek újraimplementálásában a szoftver-újratervezés folyamata segíthet karbantarthatóbbá tenni ezeket. Az újratervezésbe beletartozhat a rendszer újradokumentálása, a rendszer újrastrukturálása és újraszervezése, a rendszer lefordítása egy modernebb programozási nyelvre, illetve a rendszeradatok struktúrájának, valamint értékeinek a módosítása és frissítése. A szoftver funkcionalitása nem változik, és általában a rendszer-architektúrája is ugyanaz marad.

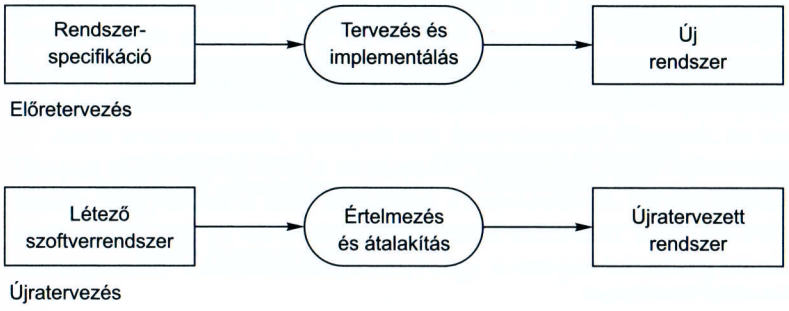
A szoftverrendszer újratervezése két fő előnnyel bír a rendszer-evolúció radikálisabb megoldásaival szemben:

1. *Kisebb kockázat.* A szervezet kulcsfontosságú rendszereinek újrafejlesztése magas kockázattal jár. Hibák kerülhetnek a rendszer-specifikációba, fejlesztési problémák merülhetnek fel stb. Az új szoftver késedelmes bevezetése üzletvesztést és extra költségeket eredményezhet. Például 1999-ben egy nagy amerikai élelmiszer-ipari cég késett az új megrendelőrendszer bevezetésével, ami miatt késtek a szállítások. Mindez végül 100 millió dolláros veszteséget okozott a csúcsszezonban.
2. *Kisebb költség.* Az újratervezés költsége sokkal alacsonyabb, mint egy új szoftver fejlesztésének költsége. Ulrich (1990) közölt egy példát, amelyben egy kereskedelmi rendszer újraimplementálásának becsült költsége 50 millió dollár volt. A rendszert sikeresen újratervezték 12 millió dollár költséggel. Ha ezek a számok mintaértékűek, az azt jelenti, hogy körülbelül négyszer olcsóbb újratervezni, mint újraírni a rendszert.

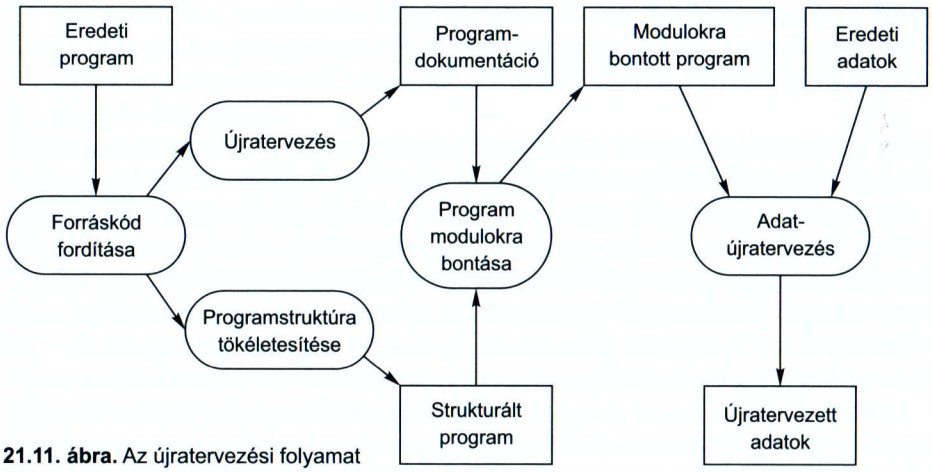
A kritikus különbség az újratervezés és a szoftverfejlesztés között a fejlesztés kezdőpontja. Ahelyett, hogy egy új specifikációt írnánk, a régi rendszer tekinthető az új rendszer specifikációjának. Chikofsky és Cross (Chikofsky és Cross, 1990) a hagyományos fejlesztést *előretervezés*nek nevezte, hogy megkülönböztesse a szoftver-újratervezéstől. Ezt a különbséget ábrázolja a 21.10. ábra. Az előretervezés a rendszer-specifikációval kezdődik, és magában foglalja az új rendszer tervezését és implementálását. Az újratervezés egy létező rendszerből indul ki, és a helyettesítő rendszer fejlesztésének folyamata az eredeti rendszer megértésén és transzformálásán alapszik.

A 21.11. ábra egy lehetséges újratervezési folyamatot ábrázol. A folyamat bemenete egy ősprogram, a kimenete ugyanannak a programnak egy strukturált, modulokra bontott verziója. A program-újratervezéssel egy időben a rendszer adatait is újratervezhetjük. Ennek az újratervezési folyamatnak a tevékenységei a következők:

1. *Forráskód fordítása.* A programot egy régi programozási nyelvről ugyanannak a nyelvnek egy modernebb verziójára vagy egy másik nyelvre konvertáljuk.
2. *Újratervezés.* A programot elemezzük, és kinyerjük belőle azokat az információkat, amelyek segítenek dokumentálni a szerkezetét és a működését.
3. *Programstruktúra tökéletesítése.* A program vezérlési struktúráját elemezzük és módosítjuk, hogy olvashatóbbá és érthetőbbé tegyük.
4. *Program modulokra bontása.* A program egymással kapcsolatban álló részeit csoportosítjuk, és ahol lehetséges, megszüntetjük a redundanciát. Néhány esetben ez a lépés architekturális változtatásokat eredményezhet, például egy centralizált egygépes rendszerből osztott platformon futó rendszer lesz.
5. *Adat-újratervezés.* Megváltoztatjuk a program által feldolgozott adatokat, hogy tükrözzék a program változásait.



21.10. ábra. Előretervezés és újratervezés

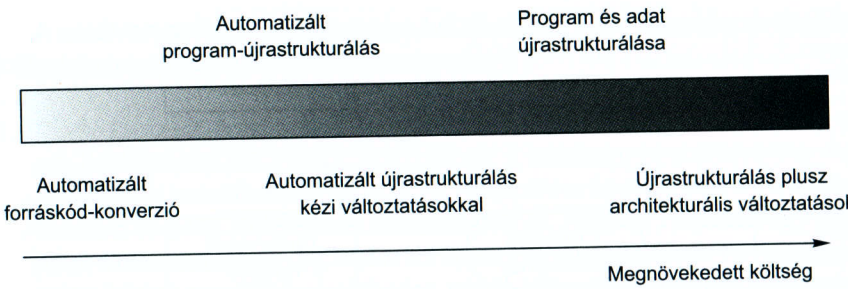


21.11. ábra. Az újratervezési folyamat

A program-újratervezés nem feltétlenül jelenti a 21.11. ábra összes lépésének végrehajtását. A forráskód fordítása nem szükséges, ha a fejlesztéshez használt programnyelvet még támogatja a fordítóprogram szállítója. Ha az újratervezés teljes mértékben automatizált eszközökön nyugszik, akkor nem szükséges a dokumentáció visszatervezés segítségével való előállítása. Az adat-újratervezés csak akkor szükséges, ha a program adatstruktúrái megváltoznak a rendszer-újratervezés során. Azonban a szoftver-újratervezés mindig a program valamilyen mértékű újrastrukturálásával jár.

Ahhoz, hogy egy újratervezett rendszer együttműködjön egy új szoftverrel, szükség lehet adapterkomponensek fejlesztésére is, mint ahogyan arról a 19. fejezetben már szó volt. Ezek elfedik a szoftverrendszer eredeti interfészeit, és új, jobban strukturált interfészeket jelenítenek meg, amelyeket már más komponensek is tudnak használni. Az ősrendszerek becsomagolásának ez a folyamata fontos technika a széles skálájú újrafelhasználható komponensek fejlesztésénél.

Az újratervezés költsége nyilvánvalóan függ a kivitelezett munka nagyságától. Az újratervezést többféle módon is megközelíthetjük, mint ahogy azt a



21.12. ábra. Újratervezési lehetőségek

21.12. ábra is mutatja. A költségek balról jobbra haladva nőnek, tehát a forráskód fordítása a legolcsóbb lehetőség, az újratervezés pedig, mint az architekturális migráció része, a legdrágább.

Az újratervezés mértékétől függetlenül, a költséget befolyásoló legfontosabb tényezők:

1. *Az újratervezendő szoftver minősége.* Minél alacsonyabb a szoftver és a hozzá tartozó dokumentáció minősége, annál magasabbak az újratervezés költségei.
2. *Az újratervezést támogató elérhető eszközök.* Normális esetben nem költséghatékony egy szoftverrendszer újratervezése, hacsak nem használhatunk CASE-eszközöket a program változtatásainak automatizálására.
3. *A szükséges adatkonverzió mértéke.* Ha az újratervezés nagy mennyiségű adat konvertálását igényli, az jelentősen megnöveli a folyamat költségeit.
4. *Az elérhető hozzáértő személyzet.* Ha a rendszer fenntartásáért felelős személyzetet nem lehet bevonni az újratervezés folyamatába, az megnöveli a költségeket. A rendszer újratervezőinek sok idejébe fog kerülni a rendszer megértése.

A szoftver-újratervezés fő hátránya, hogy gyakorlati korlátok szabnak határt az újratervezéssel elérhető javulás mértékének. Például nem lehetséges egy funkcionális megközelítéssel megírt programot objektumorientált rendszerré konvertálni. Nagyobb architekturális változtatások vagy a rendszer adatkezelésének átszervezése nem kivitelezhető automatikusan, ezért magas többletköltségekkel jár. Annak ellenére, hogy az újratervezés megnövelheti a karbantarthatóságot, lehetséges, hogy az újratervezett rendszer nem lesz olyan jól karbantartható, mint egy új, modern szoftvertervezési módszerekkel fejlesztett rendszer.

21.4. ŐSRENDSZEREK EVOLÚCIÓJA

Az új szoftverrendszereknél, amelyeket modern szoftvertervezési folyamatokkal fejlesztettek, mint például az iteratív fejlesztés és a CBSE, lehetséges a rendszerfejlesztés és az evolúció integrálásának a megtervezése. Egyre több és több cég kezdi felismerni, hogy a rendszerfejlesztés egy, a teljes életcikluson átívelő fo-

lyamat és haszontalan a szoftverfejlesztés és a karbantartás mesterséges elkülönítése. Ennek ellenére még mindig sok ősrendszer létezik, amelyek üzleti szempontból kritikusak. Ezeket a rendszereket ki kell terjeszteni, és adaptálni kell a folyamatosan változó e-business szokásokhoz.

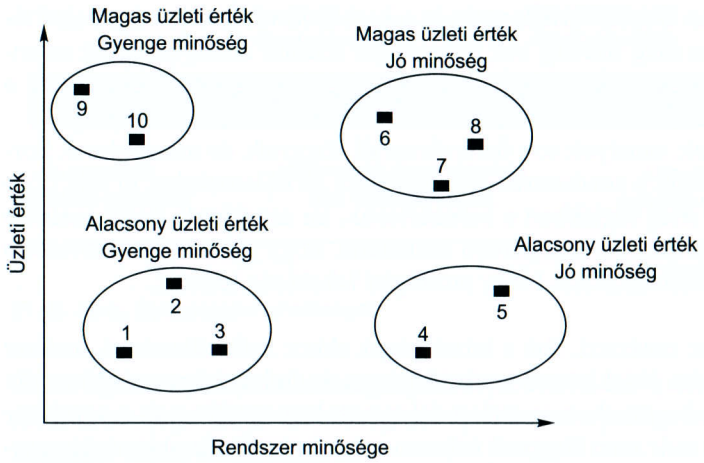
Azok a szervezetek, amelyek sok ősrendszertől függnek, és amelyeknek korlátozott a költségvetésük e rendszerek fenntartására és fejlesztésére, el kell dönteniük, hogyan térül meg legjobban a befektetésük. Ez azt jelenti, hogy reálisan értékelniük kell az ősrendszereiket, majd eldönteni, hogy mi a leghelyénvalóbb stratégia ezek továbbfejlesztésére. Négy stratégiai lehetőség létezik:

1. *Kiselejtezni az egész rendszert.* Ezt a lehetőséget akkor kell választani, amikor a rendszer már nem járul hozzá eredményesen az üzleti folyamatokhoz. Ez akkor következik be, amikor az üzleti folyamatok megváltoztak a rendszer bevezetése óta, és már nem függnek teljesen a rendszertől. Ez akkor a leggyakoribb, amikor a nagyszámítógépes terminálokat PC-kre cserélik, és ezeken a gépeken kész szoftverek biztosítják az üzleti folyamatok által igényelt számítógépes támogatást.
2. *Folyamatosan tovább fenntartani a rendszert.* Ezt a lehetőséget akkor kell választani, amikor a rendszerre – amely meglehetősen stabil – még szükség van, és a felhasználók nem igényelnek nagyszámú változtatást.
3. *Valamilyen módon megváltoztatni a rendszert, hogy javuljon a karbantarthatósága.* Ezt a lehetőséget akkor kell választani, amikor a rendszer minősége leromlott a rendszeres változtatások miatt, és ahol továbbra is szükség van rendszeres változtatásokra. Mint már említettem, ez a folyamat magában foglalhatja új interfészkomponensek kifejlesztését, így az eredeti rendszer együtt tud dolgozni más, újabb szoftverrendszerekkel.
4. *A rendszer lecserélése új rendszerre.* Ezt akkor kell választani, amikor más egyéb tényezők, mint például új hardver következtében a régi rendszer nem folytathatja működését, vagy ahol léteznek azok a kész rendszerek, amelyek elfogadható költségen lehetővé teszik az új rendszer fejlesztését. Számos esetben az evolúciós lecserélési stratégiát lehet alkalmazni, amelyben a fő rendszerkomponenseket, ahol csak lehetséges, dobozos rendszerekkel, illetve újrahasznált komponensekkel helyettesítik.

Természetesen ezek a lehetőségek nem egymást kizárók, így ahol a rendszert több különböző program alkotja, különböző lehetőségekkel élhetünk a rendszer különböző részeinél.

Amikor egy ősrendszert értékelünk, két különböző nézőpontból kell tekintenünk a rendszert (Warren, 1998). Üzleti szempontból a rendszer üzletben képviselt értékét kell értékelnünk. A rendszer nézőpontjából az alkalmazói szoftver minőségét és a rendszert támogató szoftvert és hardvert kell értékelni. Az üzleti érték és a rendszer minőségének kombinációja használható majd arra, hogy segítse a döntést, hogy mit tegyünk az ősrendszerrel.

Hogy illusztráljuk ezt az értékelést, tegyük fel, hogy egy szervezetnek 10 ősrendszere van. Minden egyes rendszer minőségét és üzleti értékét meghatároz-



21.13. ábra. Rendszerminőség és üzleti érték

tuk és összehasonlítottuk egymással közös diagramon ábrázolva a relatív üzleti értéket és rendszerminőséget. Ezt illusztrálja a 21.13. ábra.

A 21.13. ábrán láthatjuk, hogy a rendszereknek négy csoportja van:

1. *Gyenge minőség, alacsony üzleti érték.* Ezen rendszerek működésben tartása drága és az üzletbe visszakerülő érték meglehetősen kicsi. Ezeket kiselejtezésre jelöljük.
2. *Gyenge minőség, nagy üzleti érték.* Ezek a rendszerek jelentősen hozzájárulnak az üzlethez, így nem lehet őket kiselejtezni. Azonban a gyenge minőségük azt jelenti, hogy az üzemeltetési költségük magas, így ezeket újratervezésre, vagy ha létezik megfelelő rendszer, akkor cserére jelöljük.
3. *Jó minőség, alacsony üzleti érték.* Ezek a rendszerek nem játszanak jelentős szerepet az üzletben, de fenntartásuk nem költséges. A lecserélésük nem éri meg a kockázatot, így a normális rendszerfenntartás folytatható, vagy ki lehet selejtezni őket.
4. *Jó minőség, nagy üzleti érték.* Ezeket a rendszereket működésben kell tartani, de jó minőségük azt jelenti, hogy nem szükséges beruházni a módosításukba vagy a rendszer lecserélésébe. A normális rendszerfenntartást kell folytatni.

Ahhoz, hogy felbecsülhessük egy rendszer üzleti értékét, azonosítanunk kell a rendszer résztvevőit, mint például a rendszer végfelhasználóit, a menedzsereiket, és ezután magával a rendszerrel kapcsolatban kérdéssorozatokat kell feltennünk. Négy fő terület van, amelyről beszélni kell:

1. *A rendszer használata.* Ha egyes rendszerek használatára csak ritkán, vagy csak emberek egy kis csoportjának van szüksége, az üzleti értékük valószínűleg alacsony. Egy ősrendszernél elképzelhető, hogy olyan üzleti érdekek kielégí-

tésére tervezték, amely már vagy megváltozott, vagy léteznek más, hatékonyabb módjai.

2. *A támogatott üzleti folyamatok.* Amikor egy rendszert bevezetnek, a rendszer által kiaknázható üzleti folyamatok megtervezhetők. Azonban elképzelhető, hogy ezen folyamatoknak a megváltoztatása lehetetlen, mert az ősrendszert azokhoz már nem lehet adaptálni. Éppen ezért lehetséges, hogy a rendszernek alacsony lesz az üzleti értéke, mert új folyamatok bevezetése nem lehetséges.
3. *A rendszer megbízhatósága.* A rendszer megbízhatósága nemcsak technikai, hanem üzleti probléma is. Ha egy rendszer nem megbízható, és a problémák közvetlen hatással vannak az üzleti felhasználókra, az azt jelenti, hogy más embereket kell átcsoportosítani a problémák megoldására, így a rendszer üzleti értéke alacsony.
4. *A rendszer kimenetei.* A kulcskérdés itt az, hogy az üzlet sikeres működésében mennyire fontos a rendszer kimenete. Ha az üzlet ezektől a kimenetektől függ, akkor a rendszernek nagy az üzleti értéke. Ha azonban ezeket a kimeneteket könnyen elő lehet állítani valamilyen más úton is, vagy ritkán használják a rendszer által előállított kimeneteket, akkor az üzleti értéke valószínűleg alacsony.

Tételezzük fel, hogy egy cég utazási megrendelőrendszert készít, ahol az utazások megszervezéséért felelős alkalmazottak rendeléseket adhatnak le egy meghatározott utazási ügynöknek. A jegyeket ezután kiszállítják, majd a cég kiszámlázza. Azonban az üzleti értékek becslésénél kiderülhet, hogy az utazási megrendelések feladására ezt a rendszert nagyon kis százalékban használják. Az utazások szervezésével foglalkozó emberek viszont olcsóbbnak és kényelmesebbnek is találhatják az utazási társaságok közvetlen elérését azok weboldalain keresztül. A rendszert lehet még használni, de nincs sok értelme megtartani, mert ugyanaz a funkcionalitás más külső rendszerekkel is elérhető.

Ennek megfelelően, mondjuk egy vállalat kifejlesztett egy rendszert, amely az ügyfelek korábbi rendeléseit felügyeli, és automatikusan emlékeztetőket küld számukra a termékek újrarendelésére. Ez számunkra az ismétlődő rendelések nagy számát eredményezi, az ügyfeleket pedig megelégedéssel tölti el, mert azt érzik, hogy a szolgáltatójuk foglalkozik a szükségleteikkel. Egy ilyen rendszer kimenete nagyon fontos az üzlet szempontjából, ezért a rendszernek magas lesz az üzleti értéke.

Egy szoftverrendszer technikai szemszögből történő megvizsgálásához tekintettel kell lenni magára az alkalmazási rendszerre és a környezetére, amelyben a rendszer működik. A környezet alatt a hardvert és minden kapcsolt szolgáltató szoftvert értünk, például fordítókat és kapcsolatszerkesztőket, amelyek a rendszer karbantartásához szükségesek. A környezet azért is fontos, mert számos rendszerváltozást magának a környezetnek a változása okoz, például hardver- vagy operációsrendszer-fejlesztések.

Ha lehetséges, a környezetvizsgálat folyamatában méréseket kell végezni a rendszerről és annak karbantartó folyamatairól. Például hasznosak lehetnek azok az adatok, amelyek a rendszer hardverének és kiszolgáló szoftvereinek karban-

tartási költségét jelzik, vagy egy adott időtartam alatt megjelenő hardverhibák száma, vagy a javítások száma, esetleg a rendszer szolgáltató szoftverén végrehajtott javítások gyakorisága.

A környezet felmérésénél figyelembe veendő tényezők a 21.14. ábrán láthatók. Észrevehető, hogy a környezetnek ezek nem csak technikai tulajdonságai.

Tényező	Kérdések
Szállító stabilitása	Létezik még a szállító? Stabil a szállító pénzügyi helyzete, és valószínűnek látszik, hogy folytatja a tevékenységét? Ha a szállító már nem létezik, van valaki helyette, aki követi a rendszert?
Hibaarány	Sok jelentett hibája van a hardvernek? Szükség volt valaha a rendszer újraindítására a támogató szoftver összeomlása miatt?
Kor	Milyen öreg a hardver és a szoftver? Minél öregebb a hardver és a támogató szoftver, annál elavultabb. Jelentős gazdasági és üzleti előnnyel járhat egy modernebb rendszerre való váltás, annak ellenére, hogy még mindig helyesen működik.
Teljesítmény	Megfelelő a rendszer teljesítménye? A teljesítménybeli problémák jelentős hatással vannak a felhasználókra?
Támogatási követelmények	Milyen helyi támogatást igényel a hardver és a szoftver? Ha a támogatásokkal járó kiadások magasak, akkor meg kell fontolni a rendszer kicserélését.
Fenntartási költségek	Mennyibe kerül a hardver fenntartása és a szoftver licence? Az öregebb hardver fenntartási költsége magasabb lehet, mint egy modern rendszeré. Magas lehet a támogató szoftver évi licencdíja.
Együttműködés	Nehézségekbe ütközik a rendszert más rendszerekhez kapcsolni? Használhatók a fordítók és a többi szoftver az operációs rendszer használatban lévő verzióján? Szükség van hardveremulációra?

21.14. ábra. A környezet értékelésének tényezői

Figyelembe kell venni a hardver és szolgáltató szoftver beszállítóinak a megbízhatóságát. Ha ezek a beszállítók már nem működnek, elképzelhető, hogy a rendszereik számára már nincs karbantartási szolgáltatás.

Egy alkalmazási rendszer technikai minőségének megvizsgálásához tényezők széles skáláját kell megvizsgálni (21.15. ábra), amelyek elsődlegesen a rendszer megbízhatóságához, karbantarthatóságához és dokumentációs nehézségeihez kapcsolódnak. A rendszer minőségének megítélésében segíthet, ha kvantitatív rendszeradatokat gyűjtünk. Néhány példa az összegyűjthető kvantitatív adatokra:

1. *A kért rendszerváltoztatások száma.* A rendszerváltoztatások rontják a rendszer struktúráját és nehezítik a további változtatásokat. Minél magasabb ez az érték, annál alacsonyabb a rendszer minősége.
2. *A rendszer által használt különböző felhasználói interfészek száma.* Ez fontos tényezője az űrlapalapú rendszereknek, ahol minden egyes űrlapot külön felhasz-

nálói interfésznek tekinthetünk. Minél több különböző interfész van, annál valószínűbb, hogy inkonzisztencia és redundancia lép fel az interfészek között.

3. *A rendszer által használt adatok mennyisége.* Minél nagyobb az adatok mennyisége (az állományok száma, az adatbázis nagysága stb.), annál összetettebb a rendszer.

Tényező	Kérdések
Érthetőség	Milyen nehéz a jelenlegi rendszer forráskódjának megértése? Milyen bonyolultak a használt vezérlési szerkezetek? A változók nevei értelmesek-e és tükrözik-e a funkciójukat?
Dokumentáció	Milyen rendszer-dokumentáció áll rendelkezésünkre? A dokumentáció teljes, konzisztens és naprakész?
Adatok	Létezik a rendszernek explicit adatmodellje? Milyen mértékű az adatok többszörös tárolása különböző állományokban? A rendszer által használt adatok naprakészek és konzisztensek?
Teljesítmény	Megfelelő az alkalmazás teljesítménye? A teljesítménybeli problémák jelentős hatással vannak a felhasználókra?
Programozási nyelv	Léteznek modern fordítók a rendszer fejlesztéséhez használt nyelvhez? Használják még a nyelvet új rendszerek fejlesztéséhez?
Konfigurációkezelés	A rendszer minden részének minden verzióját egy konfigurációkezelő rendszer vezérli? Létezik explicit leírás a jelenleg használt rendszer komponenseinek verzióiról?
Tesztadatok	Léteznek tesztadatok a rendszerhez? Vannak kidolgozott regressziós tesztadatok arra az esetre, amikor egy új eszközzel bővül a rendszer?
Hozzáértő személyzet	Vannak a rendszer fenntartásában jártas szakemberek? Kevés olyan ember van, aki érti a rendszert?

21.15. ábra. Az alkalmazások értékelésének tényezői

Annak ellenére, hogy ezek hasznos adatok, az összegyűjtésük nagyon költséges lehet. Ez azt jelentheti, hogy nem praktikus az értékeléshez összegyűjteni őket. Továbbá nincsenek abszolút értékek, amelyeket használni lehetne. A rendszer korát és méretét akkor kell számításba venni, amikor a minőségi döntés méréseken alapszik.

Elméletileg objektív méréseket kellene végezni annak eldöntésére, hogy mit is csináljunk egy ősrendszerrel. Azonban számos esetben ezek a döntések nem igazán objektívek, hanem szervezési vagy politikai döntéseken alapulnak. Például két üzlet egyesülésénél a politikailag legerősebb partner tartja meg a rendszereit, és törli el a többi rendszert. Ha egy szervezet felső vezetősége az új hardverplatform bevezetése mellett dönt, akkor az alkalmazások cseréi is szükségesek lehetnek. Ha a költségvetés nem elegendő egy bizonyos évben a rendszer átalakítására, akkor a rendszer karbantartása annak ellenére folytatódhat, hogy ez hosszú távon magasabb költségeket fog eredményezni.

Kulcsfogalmak

- A szoftverfejlesztésnek és az evolúciónak egy egyszerű, integrált iteratív folyamatnak kell lennie, amely egy spirális modellel ábrázolható.
- Lehman törvényei, mint például a folyamatos változás fogalma, a rendszerevolúció hosszú távú elemzéséből származó szempontokat mutatnak be.
- A szoftverkarbantartásnak három típusa van: hibajavítás, a szoftver módosítása egy új környezetben történő működtetéshez és új vagy megváltozott követelmények implementálása.
- Átlagos rendszereknél a szoftverkarbantartás költsége meghaladja a szoftverfejlesztés költségét.
- A szoftverevolúció folyamatát a változtatási kérelmek vezérlik, ami magában foglalja a változtatás hatáselemzését, a verziótervezést és a változtatás implementálását.
- A szoftver-újratervezés a szoftver újrastrukturálásával és újradokumentálásával van kapcsolatban, hogy az érthetőbbé váljon és könnyebben megváltoztatható legyen.
- Egy ősrendszer üzleti értékét, valamint az alkalmazói rendszer és környezetének a minőségét fel kell mérni annak meghatározásához, hogy a rendszert átalakítani, karbantartani, vagy esetleg újratervezni kell-e.

További irodalom

Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices. Ez a kiváló könyv bevált eredményeket mutat be a szoftverkarbantartás, az evolúció és az ősrendszerek migrációjának területéről. Ezt a könyvet egy COBOL-rendszer Java-alapú szerver–kliens-rendszerre való transzformációjakor végzett esettanulmányra építették. (Seacord, R. C. *és mások*, 2003, Addison-Wesley.)

The Renaissance of Legacy Systems. Ez a könyv leginkább az ősrendszerek evolúcióját érinti. Mindezek mellett alapvető értekezéseket tartalmaz ezekről a rendszerekről, továbbá esettanulmányokat, amelyek az ősrendszerek struktúráit mutatják be, és egy fejezetet a rendszer felméréséről. (Warren, I., 1998, Springer.)

Feladatok

21.1. Magyarázza meg, hogy egy valós környezetben használt szoftverrendszert miért kell megváltoztatni, vagy miért lesz fokozatosan egyre használhatatlanabb.

21.2. Magyarázza el Lehman törvényeinek értelmét. Milyen körülmények között nem állják meg a helyüket ezek a törvények?

21.3. Röviden írja le a szoftverkarbantartás három típusát. Miért nehéz néha különbséget tenni közöttük?

21.4. Mint egy külföldi olajiparra specializálódott cég szoftver projektmenedzsere, azt a feladatot kapja, hogy a cége által fejlesztett rendszer karbantartási tényezőit állapítsa meg. Írja le, hogy hogyan készítene elő egy programot a karbantartási folyamat elemzésére, és adjon megfelelő karbantartási metrikákat a cégének.

21.5. A 21.7. ábra azt mutatja, hogy a hatáselemzés fontos részfolyamata a szoftverevolúciós folyamatnak. Egy diagramon ábrázolja, hogy milyen tevékenységeket tartalmazhatna a változtatás hatáselemzése.

21.6. Melyek a rendszer-újratervezés költségének alapvető tényezői?

21.7. Melyek a szükséges feltételek ahhoz, hogy sikeres legyen a szoftver-újratervezés?

21.8. Milyen körülmények között dönthet úgy egy szervezet, hogy kiselejtezi a rendszert annak ellenére, hogy a rendszerelemzés szerint magas a minősége és az üzleti értéke is?

21.9. Melyek az ősrendszerek evolúciójának stratégiai lehetőségei? Normális esetben mikor cserélné ki (szoftver-újratervezéssel vagy anélkül) a rendszer részét vagy egészét a további karbantartás helyett?

21.10. Magyarázza meg, hogy a kiszolgáló szoftver problémái miért jelenthetik a szervezet ősrendszereinek kicserélését.

21.11. Van-e a szoftverfejlesztőknek felelőssége abban, hogy olvasható kódot készítsenek akkor is, ha ez nem feltétlenül elvárás a munkaadójuktól?

21.12. Egy szervezet menedzsmentje megkéri önt, hogy készítsen felmérést a rendszerükről, és azt szeretnék, ha a felmérés eredményei igazolnák a rendszer elavultságát és egy új rendszerre történő lecserélésének szükségességét. Ez azt jelenti, hogy a rendszerkarbantartók egy része elveszíti a munkáját. Az ön kimutatása viszont azt mutatja, hogy a rendszer jól karban van tartva, jó minőségű és magas az üzleti értéke. Hogy jelentené ezeket az eredményeket a szervezet menedzsmentjének?