DIPLOMAMUNKA

Csordás Tamás

Debrecen

2009

Debreceni Egyetem

Informatikai Kar

Adatmigrációs folyamatok elemzése és implementálása

Témavezető:

Prof. Dr. Végh János

DE - Informatikai Kar

Informatikai Rendszerek és Hálózatok Tanszék

Készítette:

Csordás Tamás

Programtervező matematikus

Debrecen

2009

1. Bevezetés

Egy profitorientált cég számára sikerességének megőrzéséhez, folyamatosan követnie kell az informatika területén tapasztalható gyors fejlődést, mind a hardverek, mind a szoftverek területén. Egy korszerűtlen alkalmazás lassíthatja az üzleti folyamatokat, hátráltathatja a mindennapi munkát. Ez nem megengedhető meg céges környezetben. A technikai fejlődés eredménye, hogy egy üzleti alkalmazást körülbelül megjelenésétől számítva 5-6 évig tekintünk korszerűnek. Az idő lejártával a cégeknek dönteniük kell a használt rendszer tovább fejlesztéséről, vagy cseréjéről.

Egy új alkalmazás fejlesztése és bevezetése egy hosszabb ideig tartó folyamat. Ennek egy része az adatmigráció. A feladat, hogy az évek során felhalmozott adatokat a régi rendszerből áttöltsük az új rendszerbe. A meglévő céges adatokat nem selejtezhetjük le, nem dobhatjuk ki, jogi és cégpolitikai akadályok miatt. Ezen adatok nem helyettesíthetőek az üzlet számára.

Ugyanakkor, mivel már egy kisebb alkalmazás esetén is nagy mennyiségű, változatos információról beszélünk, az adatbetöltés – más néven adatmigráció – nem triviális feladat. A felmérések szerint az adatmigrációs projektek közel 80% lépi túl az előre eltervezett idő és költség keretet, harmaduk be sem fejeződik. Mivel az adatmigrációs egy komplexebb fejlesztés része, az ilyen jellegű csúszások a teljes projektre hatással van. Egy korszerűbb pénzügyi rendszer, vagy egy egységes CRM rendszer bevezetésekor egyszerűen nem engedhetőek meg a hibák és csúszások.

Az adatmigráció nem más, mint amikor adatokat másolunk egy olyan rendszerből, amit alig vagy egyáltalán nem ismerünk, egy olyan rendszerbe, ami még fejlesztés alatt áll, és közel sem mondható befejezettnek. Ha a célrendszer egyes részei változnak, azt a migrációs folyamatokban is követni kell.

Diplomamunkám keretében megpróbálom meghatározni a migrációs projektek buktatóit, a lehetséges hibaforrásokat illetve, azon lépéseket, amikkel ezeket el lehet kerülni. Emelet, szeretném bemutatni konkrét példákon keresztül a migrációs folyamat lépéseit és néhány hatékony eszközt, melyeket munkám során használtam.

2. Fogalmak

Adatmigrációnak nevezzük az eltérő forrásból (rendszerből) származó hasonló tartalmú adatok integrációját egy új vagy létező rendszerbe. A célrendszer sok esetben eltérő üzleti logikát valósít meg, adatstruktúrája kisebb-nagyobb mértékben eltérhet, esetleg más technológiára épül. Jellemzően újonnan bevezetett rendszerek esetén végezzük, de adatmigráció szükséges lehet szoftverek, adatbázisok verzióváltásakor, rendszerek összevonásakor, technológiai változások alkalmával is. Történhet manuálisan, de leggyakrabban automatizált folyamatként megy végbe. Költséges és erőforrás igényes, legtöbbször nagyméretű adathalmazokon végzik.

Migrációs adatoknak nevezzük azon elemeket, amik a migrálandó rendszerben vannak tárolva. Ezen adatok lehetnek adatbázis rekordok, dokumentációk, egyszerű szövegek, képek vagy bármilyen más, információt hordozó elemek. A felsorolt típusú elemek közül bármelyik migrálása szükséges lehet.

Adattisztításnak nevezzük azt a tevékenységet, amikor a migrálandó adatokat valamilyen előre meghatározott módon módosítjuk. Adattisztítás szükséges, ha

- a rendszerben hibás adatok szerepelnek,
- hibák történnek az adatbázisokból való átvétel alatt,
- bizonyos paraméterekkel rendelkező adatokra nincs szükség a migráció alatt.

Az adattisztítás történhet manuálisan, automatikusan illetve Félautomatikusan.

3. Migrációs folyamat vagy projekt?

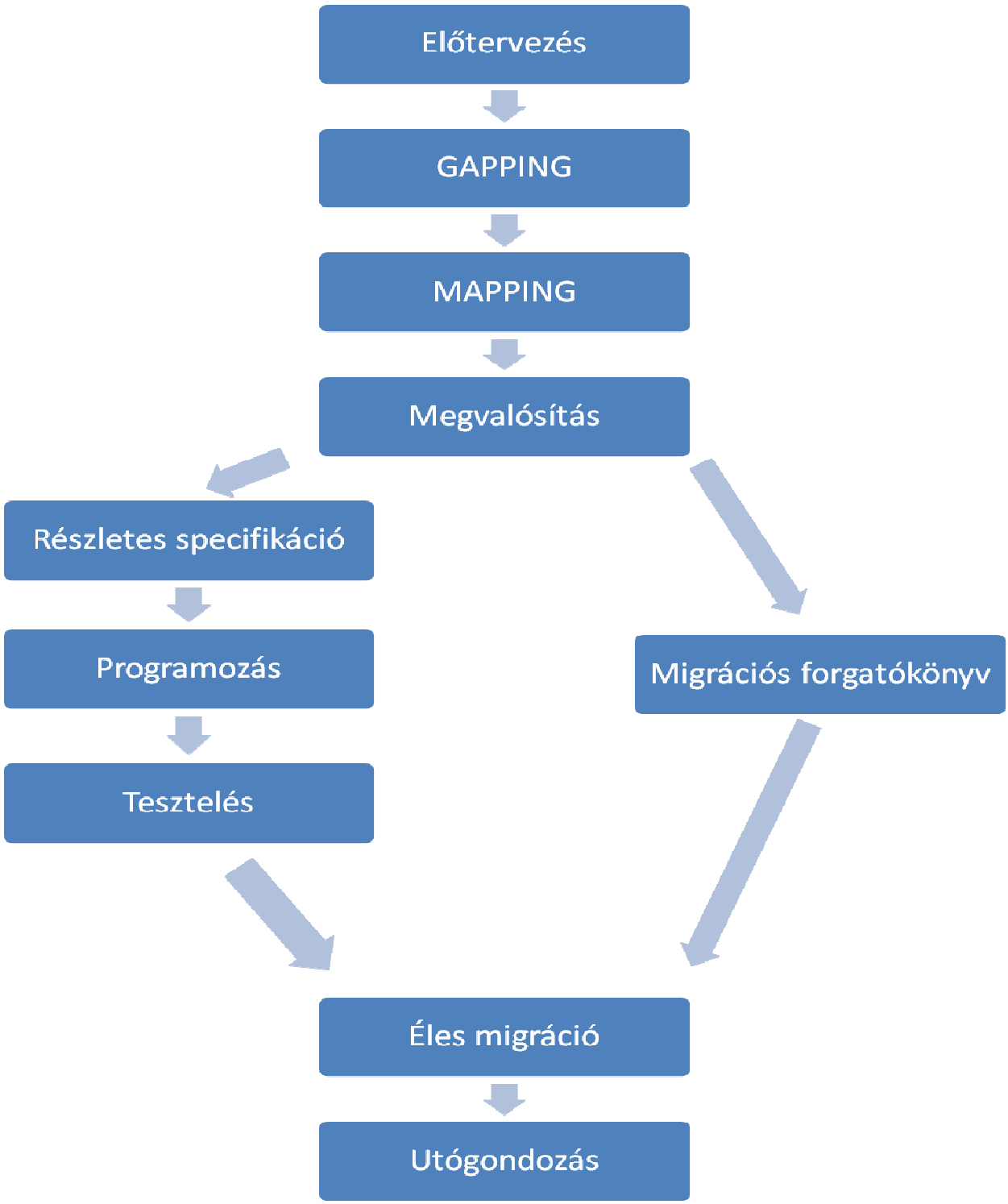
Az alkalmazásfejlesztéssel foglalkozó szervezetek között igen elterjedt, de hibás felfogás, hogy az adatmigráció egy egyszeri, bonyolult, de gyorsan elvégezhető folyamat. Ez sajnos nem igaz, mert az adatmigráció sokkal komplexebb feladat, mint az elsőre gondolnánk. A megfelelő tervezés, egyeztetés, dokumentálás, az eszközök kifejlesztése, a használt módszer meghatározása és az előforduló hibák javítása mind idő és költségigényes folyamatok. A nem megfelelő szemléletmód nagyban növeli a csúszások esélyét, növeli a költségeket, és gyakran az egész projekt befulladását is eredményezheti. Még azon cégek is, akik foglalkoztak adatmigrációval, vagy kifejezetten arra szakosodott, még nekik is gyakran igen nagy kihívást jelenthet egy nagyobb alkalmazás adatainak megfelelő módon történő áttöltése.

Ennek oka az, hogy a munka kezdetén nem teljesen meghatározottak a követelmények és a célok. Számos potenciális kérdés vetődhet fel az adatforrásokról, a migrálandó adatokat illetően, de ezeket nem láthatjuk át addig, amíg nem állnak rendelkezésre naprakész információk, amíg nem ismerjük meg annak tartalmát, szerkezetét, az adatok minőségét és az adatok közötti összefüggéseket, kapcsolatokat. A lehetséges buktatók száma szinte végtelen, a forrás rendszer alapos megismerésével és a tervek többszöri javításával oldható meg, hogy az adatokat az elvártaknak megfelelően tudjuk használni a célrendszerben. Továbbá az adatmigrációs projektekkel szemben támasztott feltételek és eredmények nagyban különböznek a teljes fejlesztés többi komponensétől. Amíg az alkalmazás fejlesztése egy jól meghatározott, egyértelmű lépésekből álló folyamat, addig az adatmigrációs projektek csak nagyobb, pontos határokkal ritkán rendelkező és egymással összefolyó lépésekre lehet bontani. Nem ritka az olyan cég, amelynek profilja pontosan az ilyen típusú fejlesztéseket tartalmazza: kifejezetten adatmigrációs projektekre szakosodtak, biztosítva a megfelelő készségeket és a megfelelő tapasztalatot.

4. Adatmigrációs folyamat lépései

Egy adatmigrációs projektet számos jól meghatározott szakaszara tudunk bontani:

1. **Előtervezés.** A projekt definiálása, szervezeti kialakítása, migrációs stratégia és megközelítés kidolgozása, ütemezés, műszaki előírások és az alkalmazási kör meghatározása, erőforrás terv elkészítése, használható eszközök keresése, beszerzése és a részletes végrehajtási terv kidolgozása.
2. **Gapping.** A rendszerek közötti különbségek feltérképezése, üzleti követelmények meghatározása, adatok elemzése, leképzések, referenciális integritások és ellenőrző vezetői döntések meghozatala a kinyert információk alapján.
3. **Mapping**. A rendszerek megfeleltetése funkcionális és adattartalmi szempontok alapján.
4. **Megvalósítás**
    a. **Részletes specifikáció.** A Mapping termékei alapján részletes program specifikáció készítése. Tesztesetek és tesztelési forgatókönyv készítése.
    b. **Programozás**. Migrációs eszközök implementálása.
    c. **Tesztelés.** Implementált eszközök futtatása tesztkörnyezetben, teszt- vagy migrálandó adatokon. Funkcionális, modul- és regressziós tesztek futtatása a részleges vagy teljes migráción.
    d. Migrációs forgatókönyv készítése. Elemi lépésekre lebontott ütemterv elkészítése. Minden lépéshez idő és erőforrás hozzárendelése, mérföldkövek definiálása és elhelyezése. Lépésekhez végrehajtó, felelős és helyettes személyek rendelése, koordinálók kijelölése. Worst case scenario kidolgozása.
5. **Éles migráció.** A forgatókönyvben definiált lépések követése, hibák kezelése, éles üzembeállás előtt migrált adatok és minden funkció alapos ellenőrzése
6. **Utógondozás.** Migráció utáni karbantartások, javítások elvégzése, helpdesk.



4.1. ábra Adatmigrációs folyamat lépései

A 4.1-es ábra az adatmigráció fázisait mutatja. A fázisok belépési és kilépési feltételeit mérföldkövekhez kötjük, melyeket előre meghatározunk. A mérföldkövek célja az ellenőrzések és értékelések elkészítése, és az eredmények bemutatása a végfelhasználóknak.

4.1 Előtervezés

Egy projekt sikerességének egyik kulcseleme az alapos tervezés. Ez az adatmigrációs projektekre az átlagosnál jobban érvényes. A nem elég alapos, nem elég körültekintő tervezés hatványozottan csökkenti a projekt megvalósulásának esélyét.

Adatmigrációs projekt tervezését szakemberek bevonásával célszerű elvégezni. A migrációs stratégia kidolgozása, szervezeti kialakítása és a megfelelő ütemezés megszervezése gyakorlati tapasztalatokat igényel. A vizsgálandó paraméterek száma magas, és a fejlesztési projekt kezdeti szakaszában ezek néha változnak vagy egyáltalán nem is vizsgálhatók.

A siker záloga a jól definiált projekt, a résztvevők körének meghatározása.

Résztvevők:

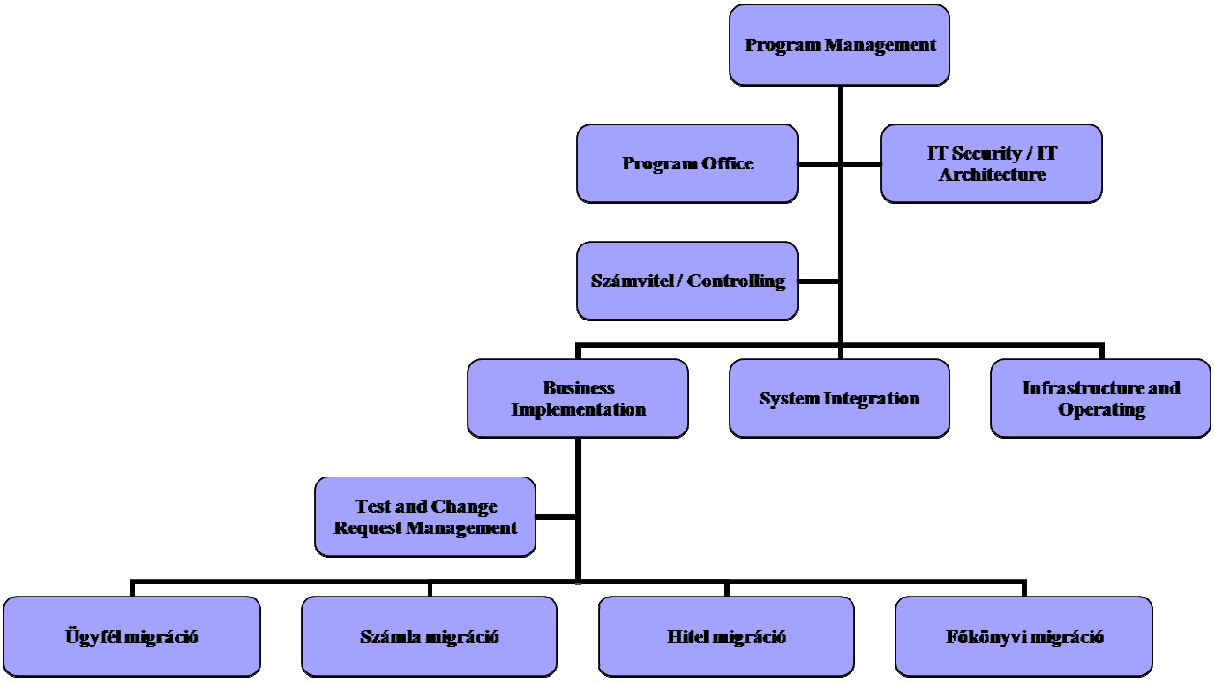
- **Projekt szponzor.** Feladata a döntések meghozatala kritikus kérdésekben. Ő biztosítja a fejlesztéshez szükséges pénzt, melyért cserébe folyamatosan tájékoztatják a projekt állapotáról.
- **Projekt koordinátor.** Feladata a projekt szponzoraival való kapcsolattartás és a fejlesztés folyamatos monitorozása. Legalább heti tájékoztatást kap a fejlesztő csapattól, kritikus döntések meghozatalára jogosult lehet. Ezen döntések okait és következményeit publikálhatja a szponzoroknak. Rajta keresztül tartja a kapcsolatot az üzleti terület és a fejlesztői csapat.
- **Projekt team.** Feladatuk a projekt megvalósítása. Főképp programozókból, adatbázis szakemberekből áll. A felmerült problémákról egyeztetniük kell az üzleti terület képviselőivel. Ha tudnak, akkor megoldási javaslatokat adnak a problémákra.
- **Üzleti terület.** A projekt teammel folyamatosan tartják a kapcsolatot. Jelzik a hibákat, észrevételeket és javaslatokat adnak a fejlesztés folyamán. Közülük kerülnek ki a későbbi felhasználók.

- **Tesztmenedzsment.** Feladatuk a migráció tesztelése és az adatminőség figyelése. A tapasztalt hibákat és vélt okokat publikálják a fejlesztői csapatnak felel. Észrevételeket és javaslatokat is tehetnek a hatékonyabb, logikusabban működő alkalmazás érdekében.

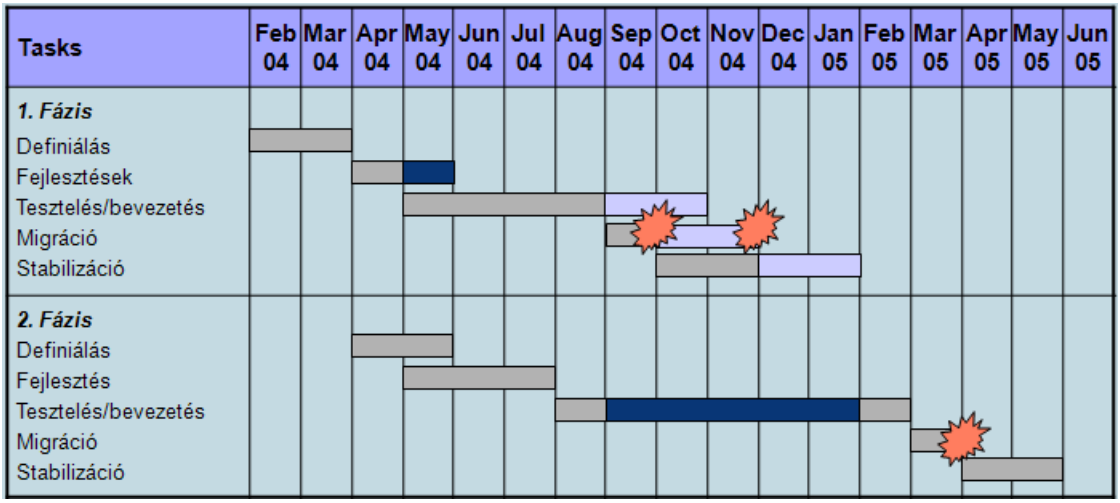
- **Minőségbiztosítás.** Jelenlétükkel lehet biztosítani, hogy a fejlesztés, és a végső alkalmazás elérje a kívánt minőséget.

4.2. ábra Projekt szervezeti felépítése

A sikeres tervezés eredményeként egy megvalósíthatósági tanulmány kidolgozására kerül sor. A dokumentum specifikálja a projekt célját, a várt eredményeket, a szerkezeti felépítést, a lehetséges megvalósítás módját és a megvalósításért felelős személyeket. Tartalmaz egy időtervet mérföldkövekkel, definiálja a mérföldköveket.



4.3. ábra Projekt szervezeti diagram



4.4. ábra Projekt terv

A megvalósíthatósági tanulmány meghatározza a megvalósítás módját és a használni kívánt eszközöket. Az adatmigrációt nagy mennyiségű, komplex adathalmazon végezzük el, ebből következően az automatikus megoldásokat részesítjük előnyben. Némely esetben viszont ez túl költséges és lassú folyamat, a manuális vagy a félautomatikus megvalósítás hatékonyabb. Mindhárom megoldási módnak több előnye és több hátránya is van.

Automatikus megvalósítás

Előnyök:

- Gyors futás
- Rugalmas, több lehetőség a hibajavításra
- Eredményei jobban tesztelhetők
- Futási ideje csak lineárisan függ az adatmennyiségtől
- Jól dokumentálható, eredménye jobban mérhető
- Hibajelzésre hatékonyabb, a keletkezett hibák jobban visszakereshetőek

Hátrányok:

- Hosszú fejlesztési időszak
- Képzett munkaerőt igényel
- Nagy az erőforrás igénye, drága eszközökkel dolgozik
- Hiba lehetősége magas

Manuális megvalósítás

Előnyök:

- Erőforrás igénye alacsony
- Szakképzetlen munkaerővel megoldható
- Az új rendszer tesztelésére és mérésére alkalmas, jó visszajelzési lehetőséget biztosít
- Adattisztítás és adatjavítás hatékonyabb

Hátrányok:

- A folyamat hosszú ideig tart

- Emberi hibázás kockázata magas
- Kevésbé dokumentálható
- Rugalmatlan

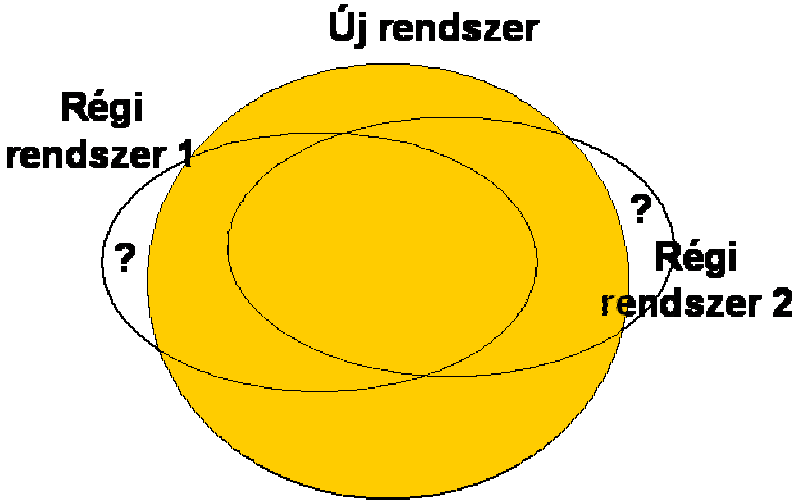
Félautomatikus megvalósítás

Előnyei és hátrányai az előbbi két megvalósítás keveréke. Félautomatikus megvalósítás alatt, azt a módot értjük, amikor a törekszünk az automatikus megoldásokra mindaddig, amíg a befektetett munka nem halad meg egy bizonyos határt. Azon problémák és feladatok megoldását, amelyek automatikus megoldása már túl költséges, azt manuálisan, emberi erő bevonásával végezzük.

4.2 Gapping

Az adatmigráció feladata a két rendszer közötti adatáttöltés véghezvitele. Ehhez alaposan ismerni kell mind a forrás, mind a célrendszert. Listázni kell, hogy a régi és az új rendszer mennyiben fedik le egymást funkcionális és adattartalmi szinten. A gyakorlatban, egy új alkalmazás bevezetésekor megpróbáljuk elkerülni a korábbi rendszer hibáit, de szeretnénk megtartani az ott már jól működő funkciókat és elveket. Mindemellett alkalmazkodni kell a technikai fejlődéshez, ha lehet mindig a „divatos”, de jól kiforrott eszközöket és elveket felhasználva fejlesztjük az új alkalmazást. Természetesen, így a két rendszer mind funkcionalitásban, mind adatszerkezetében, kisebb-nagyobb mértékben, de eltér.

A rendszerek közötti különbségek feltérképezését az új rendszert fejlesztő csapat végzi el, de nélkülözhetetlen a korábbi fejlesztők, és az üzleti felhasználók segítsége. A vizsgálat jelentős erőforrásokat igényel. Egy egyszerűbb szerkezeti vizsgálat, illetve néhány kiragadott mintaadat nem szolgáltat elég átfogó képet a migrálandó rendszerről, a rendszerben lévő adatok összefüggéseiről, és minőségéről.



4.5. ábra Rendszerek funkciók térképe

Az analízis végeztével több olyan döntést kell meghozni, mely a későbbi munkát nagyban befolyásolja. Éppen ezért ezen döntéseket a projekt szponzorainak és a koordinátoroknak kell meghozni. Ilyen kérdések például:

- Az új rendszer bővítése szükséges-e a régi rendszer funkcióinak megőrzése melett, vagy:
- lemondunk a funkcionalitásról?
- Az új rendszer működéséhez szükséges, de korábban nem létező ilyen típusú adatokat, hogyan biztosítsuk.
- Mit tegyünk a „felesleges” adatokkal?
- A helytelen adatok javítását ki és milyen módon végzi?
- Mi a teendő a nem javítható adatokkal?

A feltérképezés optimális esetben a kívánt módon elvégezhető. Gyakori azonban, hogy üzletpolitikai vagy emberi tényezők miatt ez csak hiányosan végezhető el. Ilyen probléma léphet fel például, ha a forrás rendszer és a hozzá kötődő adatbázisok titkosak, jogi vagy egyéb akadályai vannak annak, hogy a fejlesztő csapat azt megvizsgálja. Ebbe érdemes az átlagosnál több időt és erőt befektetni, hiszen az emiatt, később előforduló koncepcionális hibák javítása túl nagy erőforrást kötve le.

A fázis feladata a funkcionális különbségek feltárása mellett a forrás rendszer adatszerkezetének feltérképezése is. A vizsgálat alatt a következő tipikus kérdésekre keressük a választ:

- Milyen típusú és állapotú a forrás adatbázis?
- Hány adatbázist kell migrálni?
- Ezek közvetlenül, vagy csak közvetve férhetőek hozzá?
- Várhatóak-e hibás adatok? És ha igen milyen mennyiségben, milyen típusúak?
- Szükséges-e adattisztítás, és ha igen, az milyen adatokat érint?
- Az adatbázis mely részére kell fókuszálni?
- Minden adatot át kell tölteni, vagy csak bizonyos feltételeknek megfelelőeket?
- Elegendő adat áll rendelkezésre?
- Mit kell tennie a strukturálatlan adatokkal?

A fázis résztvevői a project koordinátorok, üzleti felhasználók, a migrációscsapat vezető fejlesztői, és ha lehetséges a korábbi alkalmazás fejlesztői.

4.3 Mapping

Miután feltérképeztük a forrás rendszer funkcionalitását, azt össze kell rendelni a célrendszer funkcióival. A funkcionális megfeleltetés sosem könnyű, a megfeleltetés gyakran nem 100%-os. A forrás- és a célrendszer üzleti logikája, a technikai megvalósítás és a felhalmozódott tapasztalatok miatt az új rendszer funkció egyszerűbbek vagy éppen bővebbek lettek. A probléma megoldása lehet a célrendszer módosítása vagy a migrálandó adatok transzformálása.

Adattartalmi megfeleltetéskor meg kell határozni, hogy az új rendszer adatait a régi rendszer mely adataiból nyerjük ki, milyen transzformációt kell alkalmazni az adatokon, a szükséges, de korábban nem létező adatokat milyen elvek alapján generáljuk, és mit teszünk adatvesztés esetén.

Az adatokon több típusú transzformációt végezhetünk. Az egyik legegyszerűbb transzformáció például a használt kódok transzformációja. A forrás és a célrendszer által hivatkozott azonosítók eltérhetnek a hosszuk, típusuk vagy éppen formájuk miatt. Ha az egyik

rendszer a rekordok azonosítására szöveges azonosítót használt, a másik pedig numerikust, akkor a migráció alatt ezen azonosítókat transzformálni kell. Sok esetben a kapott adatok szerkezete teljesen eltér a célrendszer követelményeitől. Néhány példa:

- Vezetéknév és keresztnév mező egyben, illetve külön tárolódik
- Cím (irányítószám, város, lakcím) külön, illetve egy mezőben tárolódik. Esetleg a cím még több részre van bontva (utca, ajtó, emelet)

Az ilyen eltérések sok esetben egyszerű javító scriptekkel megoldható. Az eltérő adattartalom az eltérő funkcionalitásból fakadhat. Az ilyen jellegű eltérések javítása, már bonyolultabb feladat, a javítás módja bonyolultabb, néha csak többszöri próbálkozással érhető el a várt eredmény.

4.4 Megvalósítás

A megvalósítás feladat az elkészült tervek alapján egy részletes specifikáció elkészítése, a migráció modulok implementálása, az elkészült modulok tesztelése, próba migrációk futtatása és a végleges migráció forgatókönyvének elkészítése. Az egyes lépések párhuzamosan futnak, és ha szükséges többször is végrehajthatóak.

4.4.1 Részletes specifikáció

A megfeleltetések alapján elkészítjük a program specifikációját. Definiáljuk a használandó eszközöket, meghatározzuk a programozási konvenciókat, melyeket betartva készül el az implementáció. Ennek lényeges szerepe az egységesítésre törekvésben és a minőség biztosításban van. Meghatározzuk a migrálandó alrendszereket és alrendszerek közötti kapcsolatot, függéseket. Definiáljuk a szükséges transzformációkat, validációs és ellenőrző scripteket. Teszteseteket és tesztelési forgatókönyveket készítünk, melyek alapján el tudjuk végezni a smoke és teljes teszteket. A fázis eredménye több dokumentumból áll, melyek a megvalósíthatósági dokumentumot felhasználva készülnek. A specifikációt közösen hozza létre a migrációs csapat és a projekt koordinátorok.

4.4.2 Implementáció

A kiválasztott nyelven és a meghatározott eszközök segítségével a specifikációban rögzített módon implementáljuk a migrációs modulokat. Az automatikus és Félautomatikus megvalósítás esetén a legfontosabb a migrációs modul gyorsasága és rugalmassága.

A migrációs modulok implementálása, saját tapasztalataim alapján a leggyorsabban és leghatékonyabban adatbázis közeli nyelvek segítségével valósítható meg. Ilyen nyelv például a PL/SQL programozási nyelv. A PL/SQL az Oracle által kifejlesztett, harmadik generációs imperatív nyelv, amit kifejezetten az SQL utasítások zökkenőmentés feldolgozására alkottak. Mivel az Oracle adatbázis az egyik legelterjedtebb adatbázisrendszer az üzleti szférában, a PL/SQL nyelv megfelelő szintű használta a migrációs fejlesztésekben megszokott. A migrációs modulok PL/SQL nyelven történő fejlesztése több előnnyel is jár:

- lehetséges nagymennyiségű adatok gyors mozgatása
- alacsony hálózati forgalom
- kiforrott és jól paraméterezhető eszközök
- adatok transzformációját, módosítását támogatja
- Oracle adatbázishoz kapcsolódó eszközök magas száma
- hordozható

A migrációs eszközöket 5 fő csoportba lehet sorolni tevékenységük alapján

- Adatkinyerő eszközök
- Konverziós eszközök
- Adatbetöltő eszközök
- Migráló eszközök
- Validációs eszközök

4.4.2.1 Adatkinyerés

Az eszköz feladat a forrás rendszerből, a migráció számára fontos adatok kinyerése. A kinyerést eredménye szerint két csoportba sorolhatjuk.

Az egyik megoldási lehetőség, mikor az adatokat közvetlenül egy másik adatbázisba mentjük. Ennek előnye, hogy a korábbi rendszertől csak minimálisan függünk, hiszen egy teljes körű adatkinyerés után, már az adatok lokális változatával dolgozunk. Ez a megoldás ritka, főképp az adott rendszer fejlesztésekor, verzió váltásakor használható.

A második megoldási lehetőség esetén köztes állományokat használunk az adatok tárolására. A megoldás első pillantásra bonyolultabb, és nem tűnik túl hatékonynak. Saját tapasztalatom alapján viszont a bonyolultságnak több haszna is van. Ilyen, hogy a forrás rendszerből történő adatkinyerést nem a célrendszert fejlesztő csapatnak kell végezni. Az adatkinyeréshez a rendszer igen alapos ismerete és akár több éves használati tapasztalat szükséges, ezért valóban érdemes a korábbi fejlesztőkre, vagy a helyi informatikusokra bízni.

Köztes állományok használatának másik előnye (bár nem mindig tűnik annak), hogy a használata előtt a feleknek egyeztetniük kell, hogy milyen típusú állományt, milyen oszlopba, milyen típusú, hosszú adat kerüljön, mi legyen az oszlopok nevei. Vagyis egy, mindkét fél által elfogadott egyeztető struktúrát kell létrehozni. Ezzel pontosan definiáljuk az egyes állományok szerkezetét, és a várt adatokat. Az ilyen szintű dokumentálás lassú és körülményes munka, előnye a későbbi stádiumokban jelenik meg.

Az egyeztető struktúra alapján már igen hamar elkezdhető a migrációs modulok fejlesztése, jóval azelőtt, hogy konkrét a konkrét tesztadatok elérhetővé vállnak. Ez nagyobb rendszerek esetén kedvező, a programozói gárdának több ideje lesz a fejlesztésre. Előfordulhat, hogy az előzetes munkák alapján olyan buktatókra, vagy jobb megoldási lehetőségekre bukkannak, melyek alapján módosítani kell mind a terveket, mind az eszközöket. Ha ez csak a konkrét adatok elérése után történik, abból kapkodás, és rossz minőségű program keletkezhet.

A példának okáért, nézzük azt az esetet, amikor a fejlesztés alatt azt tapasztaljuk, hogy a kiválasztott és megvásárolt adatbetöltő eszköz sebessége az adatmennyiség miatt nem megfelelő, 4-5-ször gyorsabb adatbetöltés lenne szükséges. Ez esetben felül kell vizsgálni a korábbi terveket, időt kell egy új megoldás kiválasztására. Megoldás lehet egy saját eszköz kifejlesztése, mely sokkal speciálisabb. Ezt az eszköz ki kell fejleszteni, le kell tesztelni és a használati módot és szabályokat rögzíteni kell. Ez felborítja a korábbi időterveket. Ha ezt még a migrálandó adatok megérkezése előtt vettük észre (szerencsére valóban így történt), akkor elegendő idő áll rendelkezésre.

Az állomány típus kiválasztása nem mindig triviális, sok tényező befolyásolja azt. Azt, hogy milyen típusú állományokat használhatunk egy későbbi fejezetben részletesebben is bemutatom, a használatuk előnyeivel és hátrányaival egyetemben.

4.4.2.2 Konverziós eszközök

A forrás és célrendszer adatstruktúrája kisebb-nagyobb eltéréseket tartalmazhat. A probléma azonosítása, és a javítás módjának meghatározása után adatkonverziós eszközökkel módosítjuk a migrálandó adatokat. A módosítás érintheti az adat típusát, tárolási hosszat és magát az adattartalmat is. Az interneten több adatkonverziós program is található, többségük pénzért megvásárolható, de szerepelnek köztük nyílt forráskódú eszközök is. Az ilyen típusú eszközök használata természetesen csökkentheti a migrációs folyamathoz szükséges időt, nem saját magunknak kell transzformációs eszközt fejleszteni. Viszont, mivel többségében fizetős szoftverekről beszélünk, a költségek nőnek. Azt, hogy milyen megoldást választunk, a felső vezetésnek kell eldöntetni – természetesen a megfelelő felmérések és tájékoztatás után. Egy kisebb rendszer esetén például gyakori a saját transzformációs eszköz fejlesztése.

Például, ha maradunk a korábbi példáknál és eszközöknél, a PL/SQL segítségével könnyen kifejleszthetünk egy, a köztes adatállományokat megfelelően feldolgozó eszközt. Ennek módja, hogy a köztes állományok tartalmát SQL INSERT scriptekké konvertáljuk, majd ezen scripteket futtatva az adatokat beemeljük a cél rendszer adatbázisába. Kihasználva az SQL transzformációs eszközeit (to_date(), trim(), round(), to_char, és reguláris kifejezések) egy hatékony, gyors és jól paraméterezhető eszközt kaphatunk. A korábbi példák megoldására a következő SQL scriptek jelenthetnek megoldást:

```
update t_customer t
set t.last_name = regexp_substr(upper(t.first_name),'^[A-Z]+'),
        t.first_name = regexp_replace(upper(t.first_name),'^[A-Z]+','')
where regexp_like(upper(t.first_name),'^[A-Z]+ [A-Z]+( [A-Z]+)*$');
```

Mivel a forrás és célrendszer másképp tárolja az ügyfél neveit, az előbbi egy mezőben, az utóbbi pedig két külön álló mezőben, adat transzformációra van szükségünk. A megegyezés szerint az ügyfél neveit a FIRST_NAME mezőben kapjuk meg. Ezen mező tartalmát a fenti SQL script segítségével vezeték- és keresztnévre bontjuk. A transzformáció után már migrálásra alkalmas lesz az ügyfél.

Típus konverzióra példaként a logikai értékek tárolási és használati módját szeretném felhozni. Tegyük fel, hogy a forrás rendszer a logikai értékeket szövegként tárolta, nevesített

konstansokként használta azokat. A lehetséges értékek TRUE, FALSE és UNKNOWN. Ezzel szemben a cél rendszer, mint egy hosszú numerikus értéket tárolja, a lehetséges értékek 0, 1 és NULL. Az SQL scriptek generálásakor a programnak figyelnie kell az ilyen értékekre, és automatikusan konverziót kell végrehajtania.

Természetesen a valós életben bonyolultabb problémákkal találkozhatunk, de megfelelő egyeztetése és tervezés után még a komolyabb problémák is megoldhatóak.

4.4.2.3 Adatbetöltő eszközök

Az adatok kinyeréséhez hasonlóan, az adatok betöltésekor is több módszer és eszköz áll rendelkezésünkre. Ha adatbázisok között közvetlenül végezzük az adatmozgatást, az adatkinyerő és betöltő eszköz ugyanaz. Ha köztes állományokat használunk, akkor a köztes állományokban tárolt adatok adatbázis elemekre való konverzióját kell elvégeznünk. Az adatbetöltő és konverziós eszközök nem minden esetben válnak kettő, bizonyos típusú adatkonverziót már az adatbetöltés alatt is el tudunk végezni. Ilyen automatikus konverzió az adat típusának és tárolási hosszának változása. Mivel a piac szoftverek általános megoldásokra törekednek, az adatbetöltés alatt egyedi adatkonverziók nem adhatóak meg. A korábbi példában szereplő név konverziót, már csak az adatok betöltése után tudjuk elvégezni.

A köztes adatállomány feldolgozásának több módját választhatjuk. Az egyik megoldás a korábban leírt mód, az adatok SQL scriptekké való transzformációja és betöltése.

A másik mód, valamely piaci eszköz használata. Ilyen általános célú adat importáló eszköz(ök) az EMS Data Import szoftver csomag. A szoftvernek több változata van, attól függően, hogy milyen adatbázisba történik az adatok importálása. Külön változat szolgál az Oracle, MySQL, PostgreSQL, InterBase/FireBird és DB2 adatbázisok alá. Az adatokat képes a .txt, .dbf, .csv, .xml, .xls állományokból kinyerni. Az eszköz működését egy későbbi külön fejezetben részletesebben is bemutatom.

4.4.2.4 Migráló eszközök

A migrációs folyamat lényegi része. Feladata az adatok betöltése a végleges adatbázis táblákba. A betöltés előtt ellenőrzéseket, konverziókat és javításokat végezhet. Szeparálható, de a konverziós és validációs eszközök is integrálhatóak bele. Mivel az adatbázisban már jelenlévő adatokon dolgozik, fejlesztése adatbázis közeli nyelven történik.

A migrációs eszközt érdemes több alrendszerből, modulból felépíteni. Az alrendszereket célszerű a cél rendszer felépítése alapján szervezni. A migrációs moduloknak külön, és egyben is működőképesnek kell lennie.

A modul feladat a betöltés mellett az előforduló hibák megfelelő jelzése és logolása. A célrendszerbe hibás adatok nem kerülhetnek. A hibás adatok nem várt működést eredményezhetnek, ezzel lassítva a fejlesztési folyamatot. A későbbi adattisztítást könnyítendő a lehető legrészletesebb hibaüzenetet kell logolni.

Mind a migrációs modulok, mind a modulon belüli részek kapcsolatban állnak egymással, érdemes megkülönböztetni a „sima" hibákat a generálódott hibáktól. Generálódott hibának nevezzük azt a jelenséget, amikor az adott rekord azért akad fenn a hibajelzésen, mert valamely hozzá kapcsolódó, korábban vizsgált adat nem felelt meg, migrálásakor hiba történt. Az migráció után, az adatokat javító szakembereknek a generálódott hibákra – melyek mennyisége igen magas is lehet – nem, vagy csak kevésbé kell figyelni, a valódi hiba okára koncentrálhatnak.

Emellett érdemes kategorizálni az eseményeket hibákra, figyelmezetésekre. Figyelmezetést küldhetünk például kiemelkedően nagy értékek, nem kötelező, de ajánlottan kitöltendő mezők esetén. A hibák jelzésére definiálhatunk egy hiba táblát, melybe az összes migrációs modul hiba esetén a megfelelő adatokat bemásolja.

A hibatáblának (ERROR_MSG) a következő attribútumai vannak

- **ERRORCODE**: Egy a hiba típusát meghatározó kód (a programozó számára jelentést hordoz), melynek segítségével a későbbiekben a hibák csoportosíthatók típus szerint, így akár statisztika is készíthető a különböző típusú hibák előfordulásáról.
- **ERRORMSG**: Szöveges hibaüzenet, a köztes állományokat biztosító fél számára is érthető. Információt szolgáltat a hiba mibenlétéről.
- **SUBJECT**: A hiba tárgya. Többnyire annak az adatmezőnek a tartalma, amely a hibát kiváltotta.
- **TABLENAME:** Azon köztes állománynak vagy táblának a neve, amelyben a hibás rekord előfordul.
- **CODE:** A köztes állományok rekordjainak azonosítója.

- **ERRORTYPE**: A hiba típusa. Pl.: WARNING, ERROR. Ezen mező segítségével, akár figyelmeztetéseket is küldhetünk, melyek segítségével felhívhatjuk a figyelmet pl. kiugróan magas értékekre.

A TABLENAME és a CODE mezők együttesen, rekord szintű hozzáférés biztosítanak a köztes állományokhoz, így könnyítve a munkáját.

4.4.2.5 Validációs eszközök

Az implementációs fázis legegyedibb eszköze. Ennek oka a célrendszer egyedisége, a célrendszer szabályai és felépítése alapján egyedileg kell elkészíteni. A validációs eszköz feladat a migrált adatok mennyiségi és minőségi ellenőrzése.

Mennyiségi ellenőrzés feladat összehasonlítani a migrálandó és migrált adatok számát, állomány és modul szinten. Vizsgálja, hogy a migráció elérte-e a megkövetelt eredményt, vagy túllépte a megszabott hiba határokat.

A minőségi ellenőrzés feladata, annak vizsgálata, hogy azok megfelelnek e a célrendszer üzleti logikájának és szabályainak. Ezen szabályok az egyszeri adatellenőrzéstől egészen a legbonyolultabb ellenőrzésekig terjedhetnek. Például, ellenőrizhetjük, hogy az ügyfél címe egy valós cím, a bankszámla száma megfelel-e a formai követelményeknek, vagy éppen a kapott kedvezmény kezdeti és végdátuma beleseik-e az ügyfél szerződésének kezdeti és végdátumába.

4.4.3 Tesztelés

Az adatmigrációs egyik kulcspontja a megfelelő tesztelés. Mivel nagyszámú, és változatos adatról van szó, a lehetséges adatvariációk száma magas. A tesztelők feladat a migrációs csapattól függetlenül elvégezni a funkcionális, modul és regressziós teszteket.

A funkciók folyamatos tesztje az adatmigrációtól függetlenül is nagy prioritású folyamat. A fejlesztés alatt a nem várt, helytelen működés és a hibák felderítése az alkalmazás minőségét befolyásolja. A tesztelők feladata emellett a nem logikus, bonyolult folyamatokra ésszerűbb megoldást javasolni. A fejlesztés alatt az új funkciókat a tesztelők és/vagy programozók által létrehozott tesztadatokon nézzük. A migráció folyamán a tesztadatok megváltoznak, az új és a régi funkciókat a migrált adatokkal újra, és talán még alaposabban le kell tesztelni.

A migrációs fejlesztési időszak alatt a programozók több, kisebb-nagyobb adathalmazt érintő próbamigrációt hajtanak végre. Ezen próbamigrációk eredménye alapján új hibák, nem várt működések, nem várt mellékhatások jelentkezhetnek. A migrációs fejlesztés csak akkor tekinthető véglegesnek, ha a teljes körű adatmigráció a tesztelési fázison átmegy, amikor a tesztelők hibamentesnek minősítik a migrált adatállományt.

4.4.4. Migrációs forgatókönyv

A fejlesztési időszak alatt, a többszöri próbamigráció tapasztalatai alapján definiálhatunk egy, az éles migrációra alkalmazandó forgatókönyvet. A forgatókönyv elemi lépésekre bontott ütemterv. Minden lépéshez meghatározzuk a végrehajtásért felelős személyeket. Ők végzik a tényleges migrációt, feladatuk a forgatókönyvben definiált lépések betartása és az esetlegesen előforduló hibák okainak felderítése, javítása. Minden lépéshez tartozik egy döntéshozó is. Az ő feladat eredményesnek vagy eredménytelennek nyilvánítani a lépést, majd a ez alapján kiadni a megfelelő utasításokat. Természetesen minden szereplő helyére egy megfelelő helyettest, aki azonnal munkára fogható, ha ez szükséges.

A lépésekhez meg kell határozni a Worst case scenario-t, vagyis azt a tervet ami a lépés sikertelensége esetén követni kell. A lépésekhez hozzá kell rendelni azt az időpontot amit túllépve végre kell hajtani a visszaállást.

Az éles migrációhoz koordináló személyek kijelölése is elengedhetetlen. Nekik van joguk a halaszthatatlan döntések meghozatalára és a migrációs lépések megfelelő összehangolása.

Mint látható az éles adatmigráció egy szigorúan szabályozott folyamat, lépések szigorú sorrendje. Minden lépésnél rögzítve van a helyes és helytelen működésre adott válasz. Az éles migráció ilyen minőségű végrehatásához a több próbamigrációs végrehajtása szükséges.

Üzleti környezetben egy rendszer leállás, vagy nem megfelelő működés súlyos anyagi következményekkel járhat. Annak megoldása, hogy a migráció előtti adatokat a lehető leggyorsabban tudjuk integrálni az új rendszerbe, hiba esetén egyből és hiba nélkül tudjunk visszaállni, és hogy az adatmigráció miatti leállás minimális idejű legyen alapos tervezést és profi végrehajtást követel. A migrációs forgatókönyv szerepe ezért ilyen fontos.

4.5 Migráció

Az éles migráció a migrációs folyamat üzleti szempontból legfontosabb szakasza. A migrációs forgatókönyv lépéseit követve végrehajtjuk az adatmigrációt éles környezetben. A megoldástól függően a korábbi rendszer leállítjuk, működéseit hosszabb rövidebb ideig szüneteltetjük. Az éles migráció előfeltétele egy friss migrációs adathalmaz, melyen a próbamigrációk többször hiba nélkül lefutottak. Szükséges egy megfelelő hatáskörrel rendelkező stáb felállítása, akik probléma esetén jogosultak a kritikus döntések gyors meghozatalára. A migrált adatok mind az éles üzembeállítás előtt és után alapos ellenőrzésen esnek át. Megvizsgáljuk, hogy a migrált adatok megfelelőek-e, tartalmaznak-e éles üzemi működést akadályozó hibát.

Nagyobb hiba esetén a vész forgatókönyv kerül végrehajtásra, mely legtöbbször a korábbi rendszer visszaállítását jelenti.

4.6 Utógondozás

Mint minden projektnek, a migrációs projektnek is van utóélete. Ennek oka többek között, hogy a felhasználók még nem szoktak hozzá az új rendszerhez, nem vagy nem teljesen értik annak működését. Emellett a tesztelések mellett maradhattak fel nem tárt hibák, bug-ok a rendszerben, melyeket későbbi patch segítségével javítani kell. Az additív fejlesztések, a rendszer további fejlesztése, újabb funkciók bevezetése szintén megköveteli az utógondozást.

A migráció utáni helpdesk csapat a migrációt végző kulcsemberekből áll, ők képesek átlátni a megfelelő szintén az adatstruktúrát, ők tudják értelmezni a kérdéseket és felderíteni a lehetséges hibákat.

5. Adatmigrációs eszközök

5.1 Köztes adatállományok

A migrációs folyamat alatt, ha a forrás rendszer adatai közvetlen nem, vagy csak igen nehezen tölthetőek be a cél rendszerbe, a köztes adatállományok használata lehet a megoldás. A forrás rendszer adatai a helyi szakemberek a meghatározott módon betöltik az előre definiált szerkezetű és számú állományba. Az állományokat eljutatják a migrációs végző csapatnak,

akik azt a meghatározott módon betöltik a célrendszerbe. Persze a folyamat nem ilyen egyszerű.

Megtalálni a két rendszer számára megfelelő állomány típust, úgy megtervezni az állomány szerkezetét, hogy az mindenkinek megfeleljen, és biztosítani, hogy a megfelelő helyre a megfelelő adat kerüljön - bonyolult és sok munkát igénylő vállalkozás. Ez egyeztetést a tervezési folyamat elején el kell kezdeni, végleges formáját pedig a valódi fejlesztés megkezdése előtt meg kell kapni. Természetesen a fejlesztés alatt kisebb-nagyobb változásokat végre lehet hajtani, bár általában már ez is sok plusz munkát jelent.

A köztes állomány típusának megfelelő kiválasztása nagyban tudja gyorsítani vagy éppen lassítani az adatmigrációt. Mivel az adatokat a köztes állományokban tároljuk a következő tulajdonságokat kell figyelembe venni:

- hordozható
- kisméretű
- szerkezete jól definiált
- nagymennyiségű adatok tárolására alkalmas
- támogatja a gyors adatkezelési műveleteket
- rugalmas

Nézzünk néhány fájl típust, mely alkalmas lehet köztes állományokhoz, és vizsgáljuk meg őket a felsorolt tulajdonságok alapján.

- **.txt**
  Az egyik legegyszerűbb fájl formátum. Szövegek tárolására alkalmas, minimális szövegformázást ismer (félkövér és dőlt betűk). Szerkezete pontosan nem definiált, de könnyen szerkeszthető és olvasható. Nagy mennyiségű adat tárolására alkalmas. Karakterkódolása beállítható, az alapértelmezett karakterkódolást a számítógép területi beállítása alapján határozza meg, de ez felülbírálható. Bonyolultabb adatok tárolására nem célszerű alkalmazni, de megfelelő elválasztó jelekkel mégis megoldható. Abból adódóan, hogy ez a formátum csak szöveg tárolására alkalmas, az adatok betöltésénél

és kinyerésénél is adatkonverziók szükségesek (dátum, szám, logikai). Ez a kétszeri adatkonverzió nagy mennyiségű adatnál már nem elhanyagolhatóan lassíthatja a migráció folyamatot, nem megfelelő minőségű adatoknál pedig hibákat eredményezhet. A lenti példában az ügyfelek adatait a customer.txt állományban tároljuk. A mezőket tabulátor választja el.

code	name	zip	settlement	address_line	bank
1354	Kovács István	4032	Debrecen	Kossuth út 32.	OTP
1355	Nagy Bence	4700	Mátészalka	Széchényi út III/2	Erste
1366	Vági Péter	4032	Debrecen	Csapó út 4/a	OTP

- **.csv**

A .csv (Comma separated values) formátum egy gyakran használt adatcsere formátum. A .csv fájlt digitális adatok táblában történő tárolására alkalmazzák. A fájl minden sora megfelel a táblázat egy sorának, a mezők vesszővel vannak elválasztva egymástól, és minden mező a táblázat egy oszlopának felel meg. Csak szöveg tárolására alkalmas, egyéb értékek használatakor adatkonverzió szükséges (az adott dátum vagy szám, mint szöveg tárolódik). A legtöbb táblázatkezelő szoftver képes beolvasnia .csv fájlt illetve képes elmenteni az adatokat ebben a formátumban. A .csv formátum régi, gyakran használt formátum: az összes számítógépes platformon használható. A .csv formátum előnye, hogy lehetővé teszi az adatok mozgatását különböző platformok között. A számítógép-tudományban, az ilyen típusú formátumokat „flat file”-nak nevezik, hiszen egy állományban csak egy táblázat tárolódik. A formátum nagy mennyiségű adat tárolására alkalmas, könnyű szerkeszteni. Egyszerűsége miatt tartalmát egyszerűen lehet módosítani. Viszonylag rugalmas, új oszlop felvétele, vagy régi oszlop törlése nem tartozik a bonyolultabb műveletek közé. A .csv formátumnak is több változata ismert, melyek különféle módon bővítik a szerkezetet. A lenti példában az ügyfelek adatait a customer.csv állományban tároljuk.

code,name,zip,settlement,address_line,bank

1354,Kovács István,4032,Debrecen,Kossuth út 32.,OTP

1355,Nagy Bence,4700,Mátészalka,Szécshényi út III/2,Erste

1366,Vági Péter,4032,Debrecen,Csapó út 4/a,OTP

- **.dbf**

  A .dbf (database file) a dBase adatbázisrendszer alapvető fájl formátuma. Széles körben használják más alkalmazások is egyszerű adatok tárolására. A dBase volt az első széles körben használt adatbázis-kezelőrendszer (DBMS). A dBase adatbázisrendszer elsőként használta a fájl szerkezet leírására a fejléceket. Ezt azt jelentette, hogy a programok magából a fájlból tudták meg a fájl felépítését, nem a programozóknak kellett előre definiálni azt. Az évek során számos DBF file szerkezet jött létre, és nem mindegyik a dBase rendszerhez kapcsolódóan. Ennek oka pontosan annak gyors elterjedése volt. Az egyes változatok nem feltétlen kompatibilisek egymással.

  A .dbf fájlok két részből állnak, a fejlécből és az adat részből. A fejléc definiálja a szerkezetet és hordozza az összes szükséges információt a fájlról(karakterkódolás, utolsó változtatás időpontja, tárolt rekordok száma, egy rekord hossza, stb.). A .dbf formátum már több adattípust különböztet meg. Többek között használhatunk szöveget, számot, dátumot, valós számot és logikai értékeket. Azt, hogy melyik oszlop, milyen típusú értéket tárolt a fejlécben tároljuk egy karakternyi helyen. Például szöveg esetén C, egész szám esetén N, logikai típus esetén egy L karakter jelöli a típust. A fejléc mérete limitált, az oszlopok nevei szabvány szerint nem haladhatják meg a 10 karaktert, de gyakorlatban a 7-8 karakter az optimális. Az eddigi fájl formátumoktól eltérően, a .dbf fájlok nem szerkeszthetőek és nézhetőek sima szövegszerkesztővel. Több .dbf olvasó és szerkesztő eszköz van forgalomban, ingyenes vagy fizetős. Ilyen eszköz például a DBF Manager 1.43 és a DBF Viewer 2000. Mivel egy elég régi és elterjedt formátumról beszélünk sok olyan eszköz található a piacon, mely segítségével a DBF fájlokat más formátumú fájlokra konvertáljuk. Ilyen gyakori konvertálások:

  - DBF to XLS
  - XLS to DBF

- DBF to XML
- DBF to CSV
- DBF to PDB(Palm DataBase)
- DBF to MDB(Access)
- DBF to SQL
- DBF to HTML
- DBF to DBF

A felsorolásból látszik, hogy hány féleképpen lehet felhasználni a DBF formátumban kapott adatokat. Olyan helyzetekben, mint a mini példaalkalmazás, ahol egy közel 10 éve kifejlesztett, dBase alapú adatbázisrendszerre épülő alkalmazás migrációját kell elvégezni, jó megoldásnak tűnik a DBF-ek használata. A forrás rendszerből a legkevesebb munkával így tudjuk kinyerni az adatokat, majd azt átadva a migrációs csapatnak többféleképp is feldolgozhatjuk azt. A felsoroltak közül személyes tapasztalatom a DBF to CSV és a DBF to SQL módokkal van. A CSV konvertálás a korábban sok problémát okozó karakterkódolást oldotta meg. A kapott DBF állományok nem egységes kódtáblát használtak, tagvállalatokon és még alrendszereken belül is több változat volt. A megoldást végül elvetettük és áttértünk az SQL-re való konvertálásra (végül saját eszköz kifejlesztésével, mely gyorsabb és kényelmesebb adatkonverzióra volt képes).

A felsorolásból érdekességként kiemelném a DBF to DBF eszközt, mely feladat a különböző típusú struktúrák közötti konverzió.

Hogy szemléltessem a DBF fájlok tulajdonságait, a lenti ábrával bemutatom a DBF fájl szerkezetét. A fejlécet az átláthatóság kedvéért két részre bontottuk, az első táblázat a fájl általános adatait, a második táblázat pedig az oszlopok leírást tartalmazza.

DBF File Header

Byte offset	Description
0	File type: 0x02 FoxBASE 0x03 FoxBASE+/Dbase III plus, no memo 0x30 Visual FoxPro 0x31 Visual FoxPro, autoincrement enabled 0x32 Visual FoxPro with field type Varchar or Varbinary 0x43 dBASE IV SQL table files, no memo 0x63 dBASE IV SQL system files, no memo 0x83 FoxBASE+/dBASE III PLUS, with memo 0x8B dBASE IV with memo 0xCB dBASE IV SQL table files, with memo 0xF5 FoxPro 2.x (or earlier) with memo 0xE5 HiPer-Six format with SMT memo file 0xFB FoxBASE
1 - 3	Last update (YYMMDD)
4 – 7	Number of records in file
8 – 9	Position of first data record
10 – 11	Length of one data record, including delete flag
12 – 27	Reserved
28	Table flags: 0x01 file has a structural .cdx 0x02 file has a Memo field 0x04 file is a database (.dbc) This byte can contain the sum of any of the above values. For example, the value 0x03 indicates the table has a structural .cdx and a Memo field.
29	Code page mark
30 – 31	Reserved, contains 0x00
32 – n	Field subrecords The number of fields determines the number of field subrecords. One field subrecord exists for each field in the table.
n+1	Header record terminator (0x0D)
n+2 to n+264	A 263-byte range that contains the backlink, which is the relative path of an associated database (.dbc) file, information. If the first byte is 0x00, the file is not associated with a database. Therefore, database files always contain 0x00.

Field Subrecords Structure

Byte offset	Description
0 – 10	Field name with a maximum of 10 characters. If less than 10, it is padded with null characters (0x00).
11	Field type: C – Character Y – Currency N – Numeric F – Float D – Date T – DateTime B – Double I – Integer L – Logical M – Memo G – General C – Character (binary) M – Memo (binary) P – Picture
12 – 15	Displacement of field in record
16	Length of field (in bytes)
17	Number of decimal places
18	Field flags: 0x01 System Column (not visible to user) 0x02 Column can store null values 0x04 Binary column (for CHAR and MEMO only) 0x06 (0x02+0x04) When a field is NULL and binary (Integer, Currency, and Character/Memo fields) 0x0C Column is autoincrementing
19 - 22	Value of autoincrement Next value
23	Value of autoincrement Step value
24 – 31	Reserved

5.1. ábra DBF fájlok szerkezete

- **.xml**

  Az XML(Extensible Markup Language) a W3C által ajánlott általános célú leíró nyelv. Az elsődleges célja strukturált szöveg és információ megosztása az Interneten keresztül. Az XML-en alapuló nyelvek formális módon vannak leírva, így lehetővé téve a programok számára a dokumentumok módosítását és validálását a formátum előzetes ismerete nélkül. Az XML, mint köztes állomány használata több előnnyel és hátránnyal jár.

  Előnye természetesen a jól definiáltsága. Szigorú szintaktikus és elemzési követelményeket támaszt, ami biztosítja, hogy a szükséges elemzési algoritmus egyszerű, hatékony és ellentmondásmentes maradjon. Több karakterkódolást is támogat, megválasztható az ideális kódolás. Olvashatóság szempontjából, ami a hibák keresésénél és az adatok minőségbeli vizsgálatánál (szemmel gyorsan átfutni, hogy milyen adatokat is kaptunk) fontos, nem a legelőnyösebb, a korábbi típusok (.txt, .csv, .dbf) erre jobban alkalamasabbak. Öndokumentáló formátum, ami külön hasznos. Egyszerű szöveg formátumban valósul meg, szerkesztése nem követel bonyolult eszközöket. Platformfüggetlen, így viszonylag immúnis a technológiai változásokkal szemben. Használata kiforrott, kész eszközök vannak, melyekkel nagyobb erőfeszítés nélkül írható és olvasható az XML állomány.

  Hátránya pontosan a jól definiáltságából következik, szintaxisa elég bőbeszédű és részben redundáns. Nagyobb adatmennyiség esetén a tárolási költség magas. Ez nehezíti a gyors feldolgozást, olvasása nagyobb memória és időigényű. Hordozhatóságát szintén befolyásolhatja a nagy méret, bár ezt bizonyos ezt a problémát a tömörítés csökkenti. Nagyobb probléma, hogy az adat típusok közül csak keveset támogat, és ezért feldolgozáskor plusz adatkonverziók szükségesek. Szintén probléma, hogy nincs lehetőség a dokumentum egyes részének közvetlen elérésére és frissítésére.

  A lenti példában az ügyfelek adatait a customer.xml állományban tároljuk.

```
<?xml version="1.0" encoding="UTF-8"?>
<Ügyfelek>
	<Ügyfél>
		<code>1354</code>
		<name>Kovács István</name>
		<zip>4032</name>
		<settlement>Debrecen</name>
		<address_line>Kossuth út 32.</name>
		<bank>OTP</bank>
	</Ügyfél>
	<Ügyfél>
		<code>1355</code>
		<name>Nagy Bence</name>
		<zip>4700</name>
		<settlement>Mátészalka</name>
		<address_line>Széchényi út III/2</name>
		<bank>Ertse</bank>
	</Ügyfél>
	<Ügyfél>
		<code>1366</code>
		<name>Vági Péter</name>
		<zip>4032</name>
		<settlement>Debrecen</name>
		<address_line>Csapó út 4/a</name>
```

<bank>OTP</bank>

</Ügyfél>

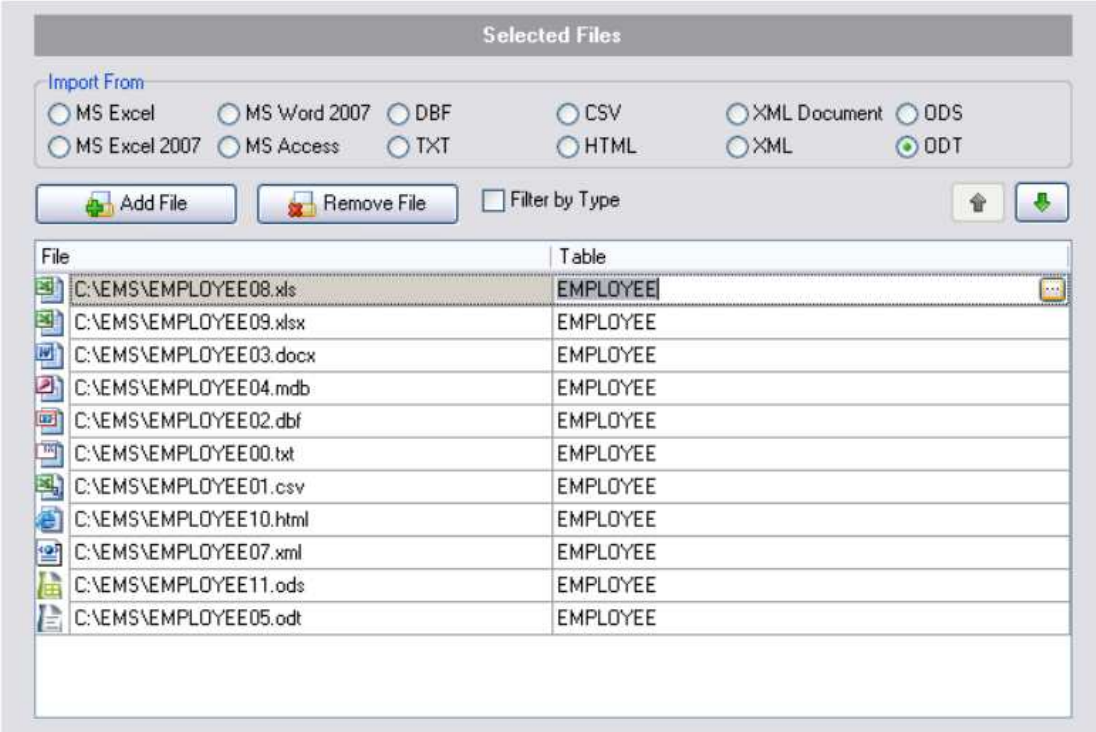
</Ügyfelek>

5.2 Adatbetöltő

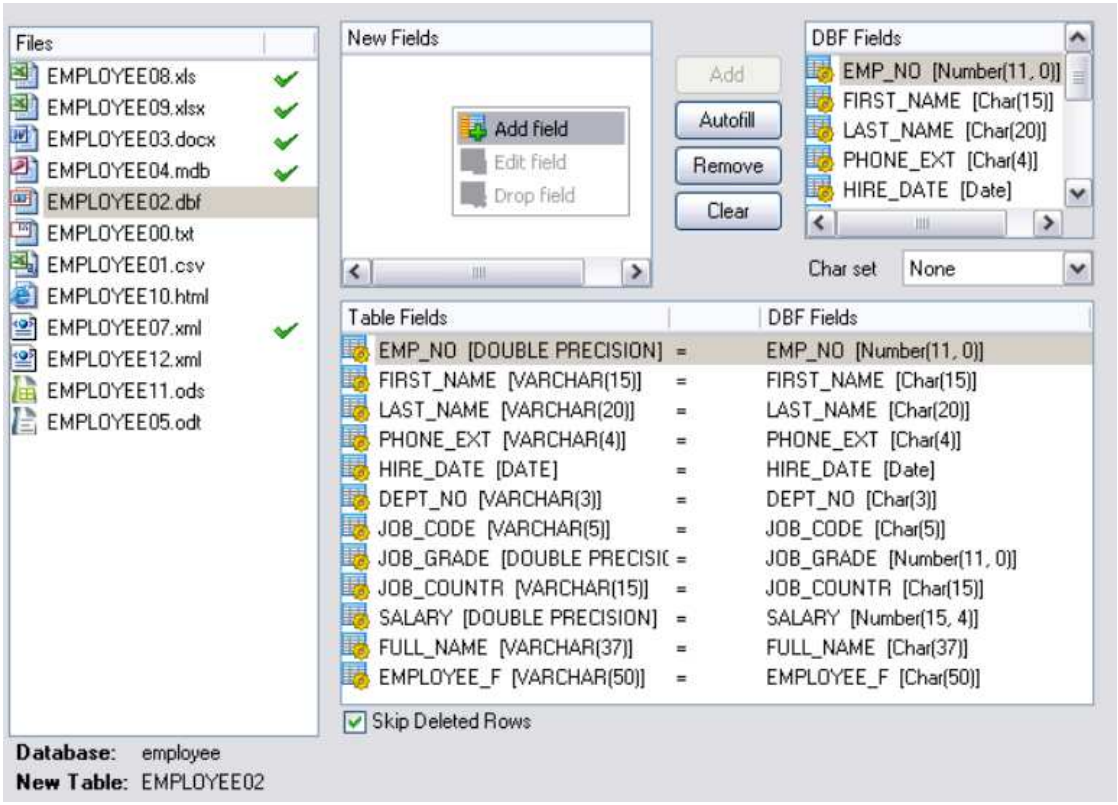
A migráció folyamán, ha szükségünk van valamilyen köztes adatállományokra, akkor alaposan meg kell fontolni, hogy mely típust választjuk. Minden formátumnak van előnye és hátránya is. A megfelelő állomány típus meghatározza az adatkinyerés és betöltés módját is. Az adatbetöltést végezhetjük külön szoftverekkel, vagy egyedileg, cégen belül fejlesztet eszközökkel. Az első választás előnye természetesen az idő, hiszen nem magunknak kell akár hetek alatt kifejleszteni az adatbetöltő eszközt. Hátránya, hogy az ilyen szoftverek a legtöbb esetben fizetősek, sok esetben drágák. Emellett a vásárolt szoftver egy általános célú eszköz, nagyon ritka, hogy 100%-osan megfelel az elvárásoknak. A saját fejlesztésű eszköz ezzel szemben egyedi, a speciális igényeknek is megfelel, a feladatra szabható. Azt, hogy melyik megoldást használjuk, befolyásolja a forrás és célrendszer típusa, a projekt mérete és a költség és időkeret.

Nagyobb projektek esetén talán célszerűbb megvásárolni egy kész eszközt. Ilyen általános célú adat importáló eszköz(ök) az EMS Data Import szoftvercsomag. A szoftvernek több változata ismert, attól függően, hogy milyen adatbázisba történik az adatok importálása. Külön változat szolgál az Oracle, MySQL, PostgreSQL, InterBase/FireBird és DB2 adatbázisok alá.

Jómagam az EMS Data Import for Oracle 2007 szoftvert használtam. Segítségével MS Excel 97-2007, MS Access, DBF, XML, TXT, CSV, MS Word 2007, RTF, ODF és HTML fájlok tartalmát tudjuk betölteni az adatbázisokba. Az eszköz több paraméter beállítására ad lehetőséget, mint például az adatforrások, táblák, importálandó rekordok számának megválasztása, COMMIT utasítások elhelyezése, a migrálandó adat automatikus konverziója az Oracle által biztosított típusra (ha lehetséges), stb. Ezen paraméterek beállítása több féle módon történhet, varázsló segítségével, parancssorból, vagy sablon alapján. A program képes egyszerre több állományt kezelni, azokat különböző táblába vagy adatbázis sémába alakítani.



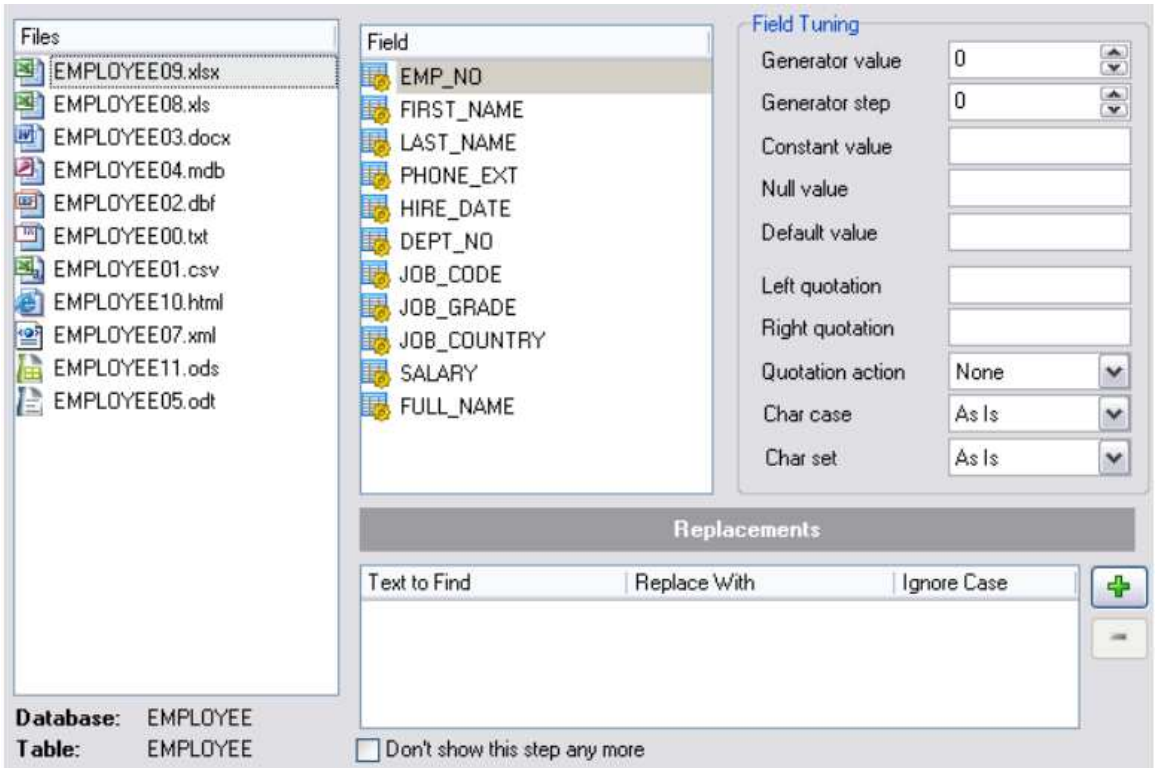
5.2. ábra Állományok kiválasztása betöltésre



5.3. ábra Oszlopok megfeleltetése

A program képes automatikusan párosítani az állomány szerkezete és az adatbázis tábla szerkezete alapján az oszlopokat. Az automatikus párosítás természetesen felülbírálható, tetszőlegesen kiválasztható a használt oszlopok, az oszlopok száma, és a betöltés helye. Az állomány minden szerkezete egy meghatározott típusúként kezelődik, a program, ha lehetséges, automatikus konverziót végez. Ha nem lehetséges, azt, mint hibát jelzi. Az 5. ábrán látható, hogy a Char(5) oszlopokat, mint Varchar2(5), a Number(15,4) oszlopokat pedig mint Double értéket tölti be az adatábázisba.

A program lehetőséget biztosít az oszlopok tartalmának bizonyos szintű módosítására is. Az oszlopokba meghatározott lépésközzel értékeket generálhatunk, megadhatunk konstans értéket, definiálhatjuk, hogy NULL érték esetén mi legyen a mező tartalma, beállíthatjuk a használt karakterkészletet, nagy- és kisbetűssé tehetjük a mező tartalmát és a megadott minta alapján cseréket hajthatunk végre.



5.4. ábra Mező beállítások

A beállításokat sémákba menthetjük, és azokat később használhatjuk, ezzel gyorsítva a migrációs folyamatot. Az adatbázis és a használt állomány típusa, az oszlopok és a műveletek

száma befolyásolja a betöltés sebességét. Tapasztalataim szerint a betöltés sebessége a 30-1500 sor/sec érték között mozog.

5.3 Adattisztító eszközök

Az adattisztítás feladata a rossz minőségű, hibás értékek nagy számban történő automatikus javítása. Az adattisztítás magában foglalja:

- az adatminőség felmérését
- javaslatot a javításra
- patch-ek írását
- programok futtatását

Mivel nagy tömegű adatokról beszélünk, melyek vizsgálatát és javítását gyorsan kell elvégezni, szintén adatbázis közeli eszközt érdemes használni. A megfelelően paraméterezett SQL scriptek alkalmasak a nagymennyiség és gyors adatjavításra.

Az első lépése az adatminőség feltárása. Ez nem más, mint az elforduló, tipikus hibák keresése, azok számának meghatározása. A leggyakoribb hiba forrása a szöveges adatok, ahol az elírás valószínűsége magas. Sajnos pontosan az ilyen típusú adatok javítása a legköltségesebb is. Emellett hibák forrása lehet az üzleti logikának nem megfelelő hibák, mint például a rossz dátumok, kiemelkedően magas számok, tévesen rögzített adatok.

A hibák feltárása után egy javaslat csomagot kell összeállítani, mely tartalmazza az egyes hibák megoldási lehetőségét, vagy ha lehetséges több megoldási lehetőséget is, illetve ezek várt következményeit. A javaslatokat egyeztetni kell a projekt szponzorával és az üzleti felhasználókkal.

A megoldásokra javító scripteket, patcheket kell készíteni, és a migrációs folyamat meghatározott fázisában futtatni. A kézi javítás nem célszerű, mert a hibázás esélye magas, és az automatizált javítással csökkenthető a befektetet munka. Másképp érdemes kezelni a tömeges és az egyedi hibákat. Míg a tömegesen megjelenő hibákra érdemes bonyolultabb scripteket írni, addig az egyedi, 1-2 alkalommal előforduló hibákat speciális, csak az adott sort érintő javító scriptet írunk. Ehhez elengedhetetlen, hogy minden egyes sort egyértelműen

meg tudjunk fogni, egyértelmű azonosítóval kell, hogy rendelkezzen. Az adattisztító scriptek futtatását külön, vagy más lépésbe integrálva is el lehet végezni.

Példa néhány SQL nyelven írt javító scriptre, az ügyfél címe több komponensre van osztva, a kapott adatok viszont egy mezőben vannak tárolva.

- Helyrajzi szám áthelyezése a megfelelő mezőbe

```
UPDATE customer c
SET c.lotNumber = regexp_substr(upper(c.addressLine), ’(H|HSZ)(\.*|:*)*( )*[0-9]+(/[0-9]+)+’),
    c.addresLine = regexp_replace(upper(c.addressLine), ’(H|HSZ)(\.*|:*)*( )*[0-9]+(/[0-9]+)+’,’’)
WHERE regexp_like(upper(c.addressLine),’(H|HSZ)(\.*|:*)*( )*[0-9]+(/[0-9]+)+’);
```

- Közterület típus áthelyezése a megfelelő mezőbe

```
UPDATE customer c
SET c.streetType =
        regexp_substr(upper(c.addressLine), ’(UTCA|ÚT|TÉR|KÖZ|SUGÁRÚT|FASOR| ….)’),
    c.addresLine =
        regexp_replace(upper(c.addressLine), ’(UTCA|ÚT|TÉR|KÖZ|SUGÁRÚT|FASOR| ….)’’’)
WHERE regexp_like(upper(c.addressLine),’(UTCA|ÚT|TÉR|KÖZ|SUGÁRÚT|FASOR| ….)’);
```

- Közterület áthelyezése a megfelelő mezőbe

```
UPDATE customer c
SET c.street = regexp_substr(upper(c.addressLine), ’^[A-Z]+( [A-Z]+\.*)*’),
    c.addresLine = regexp_replace(upper(c.addressLine), ’^[A-Z]+( [A-Z]+\.*)*’,’’)
WHERE regexp_like(upper(c.addressLine),’^[A-Z]+( [A-Z]+\.*)*’);
```

- Rövidítés javítása

```
UPDATE customer c
SET c.street = 'Móricz Zsigmond'
WHERE regexp_like(upper(c.street),'^M(O|Ó)RICZ ZS(\.)*[A-Z]*$');
```

- Hiányzó irányítószám megadása

```
UPDATE customer c
SET c.zip = '4700'
```

```
WHERE upper(c.settlement) = 'MÁTÉSZALKA';
```

- Ha az érvényes, de lejárt szerződés végdátuma nincs beállítva, akkor a végdátumot beállítjuk a lejárati dátummal, és a státuszát befejezettre állítjuk (’060’ státusz kód)

```
UPDATE contract cnt
SET     cnt.end_date = cnt.expering_date,
        cnt.status_code = '060'
WHERE   1 = 1
        AND cnt.end_date IS NULL
        AND cnt.expering_date IS NOT NULL
        AND cnt.is_validated = 1;
```

6. Összefoglalás

Diplomamunkám keretében megpróbáltam a migrációs folyamat lépéseit definiálni, megfelelő rendszerbe szerkeszteni. Az egyes lépésekben végrehajtandó műveleteket azonosítani, az előforduló hibákra figyelmeztetni, és megoldást adni a hibák elkerülésére. Mivel az elméleti összefoglalás nem elégséges, megpróbáltam szemléltetni az elvégzendő munkát néhány gyakorlati példával és a témában szerzett saját tapasztalataim leírásával.

Véleményem szerint sikerült egy megfelelő és későbbiekben is használható elméleti összefoglalót létrehoznom. A folyamat komplexitása és sokszínűsége miatt a definiált lépések felülbírálhatóak, összevonhatóak, mint például a Gapping-Mapping páros, vagy további, különálló lépésre bonthatóak.

Az elméletei áttekintés alatt saját gyakorlati tapasztalatomra alapozva és a témában megjelent cikkekre – melyek száma nem túl magas, hiszen a valódi részletek a vállalti titkok kategóriájába tartozik – hagyatkoztam.

A konkrét eszközök bemutatása során az implementációs szakaszra koncentráltam, a tervezési és vizsgálati fázis túlontúl egyedi ahhoz, hogy általánosságban tudjak róla beszélni. Egy konkrét példa bemutatása pedig túlmutat ezen dokumentum hosszán.

Összességében látható, hogy az adatmigráció az alkalmazás fejlesztés esetén egy magas priorítású tevékenység. Az alkalmazás és a mögötte lévő adatok minőségétől és mennyiségétől függően változik a komplexitása. Pontosan ezért, megfelelő mennyiségű emberi és egyéb erőforrást kell fenntartani ahhoz, hogy a kívánt minőségben és a meghatározott időn belül befejeződjön. Többszörözötten igaz rá, hogy az előzetes tapasztalatok segíthetnek a tipikus hibák elkövetésére.

A diplomamunka írása közben és a témával kapcsolatos cikkek keresése közben jöttem rá, hogy mennyire komplex is a folyamat. A vizsgálandó paraméterek száma, a folyamatos változás és a felmerülő és megoldásra váró problémák száma, mind azt sugalták, hogy egy egyszerű diplomamunka terjedelméből következően nem alkalmas a teljes folyamat, a megfelelő részletességgel történő bemutatására.

7. Köszönetnyilvánítás

Itt szeretnék köszönetet mondani témavezetőmnek, Dr. Végh Jánosnak, illetve külső témavezetőmnek, Boda Bélának, akik a dolgozat megírása közben ötletekkel, észrevételekkel segítettek abban, hogy dolgozatomat kerek egésszé formáljam. Köszönöm tanácsaikat és végtelen türelmüket.

8. Ábrajegyzék

9. Idézett forrásmunkák

Data Migartion. (dátum nélk.). Forrás: Infotechnet: http://www.infotechnet.org/ntca/DataMigration.htm

Mohanty, S. (2004). *Data Migration Strategies*. Forrás: Information management: http://www.information-management.com/specialreports/20040518/1003611-1.html

Parthasarathi, A. (2007). *Assessing Data Migration Readiness*. Forrás: Enterprise Systems: http://esj.com/articles/2007/01/23/assessing-data-migration-readiness-part-1-the-right-way-to-begin.aspx