Dr. Lengyel József

Informatikai alapok jövendő gazdasági informatikusoknak

Innovatív megoldások a WSUF hallgatói létszámának növelésére,
MTMI képzési palettájának erősítésére - EFOP-3.4.4-16-2017-00028

WEKERLE SÁNDOR ÜZLETI FŐISKOLA
BUDAPEST

INFORMATIKAI ALAPOK JÖVENDŐ GAZDASÁGI INFORMATIKUSOKNAK

Készült az EFOP-3.4.4-16-2017-00028 számú, „Innovatív megoldások a WSUF hallgatói létszámának növelésére, MTMI képzési palettájának erősítésére” című projekt keretében.

Budapest, 2017-2019.

Tartalom

1. Rendszerfejlesztési folyamat és módszerek

E téma feldolgozásához fontos, hogy Ön megismerjen alapfogalmakat, melyek segítenek a további ismeretek elsajátításában, megértésében és rendszerezésében.

A bevezető alapismereteket követően bemutatjuk a rendszerfejlesztés életciklusát (a rendszerfejlesztési szakaszokat) és a legjellemzőbb klasszikus rendszerfejlesztési életciklus modelleket. Külön tárgyaljuk a fejlesztési követelményekre vonatkozó specifikáció tartalmát, továbbá a szervezeti és technikai specifikáció előírásait.

A tanuláshoz készített tananyag következő részében megtalálja a rendszerfejlesztés felvázolt életciklus szakaszaiban használatos CASE (Computer Aided System Engineering: számítógéppel támogatott szoftver-fejlesztés) eszközöket, majd megfelelő részletességgel tárgyalja a tananyag különböző fejlesztő eszközöket (szerkesztő hibakereső, fordító, tesztelő eszközök) és azok használatát is.

A rendszerfejlesztésnek igen fontos szakasza a tesztelés és a rendszer bevezetése. Ezzel külön is kell foglalkoznia, részleteiben megismerheti a különböző teszttípusokat. A rendszerimplementációs (bevezetés) fázisban lévő feladatokat (szoftver átadás, adatmigráció, betanítás, felhasználói támogatás) megismerheti a következő részben. Végezetül Ön megismeri a legfontosabb rendszerfejlesztési dokumentumokat.

A tananyag eltérő mélységben tárgyalja az egyes résztémákat. Célunk az, hogy Ön folyamatot és összefüggéseket ismerjen meg, és megalapozzuk tudását a kapcsolódó témák tanulásához is.

1.1. Szoftver az adatfeldolgozó rendszerekben

Tanulási célok

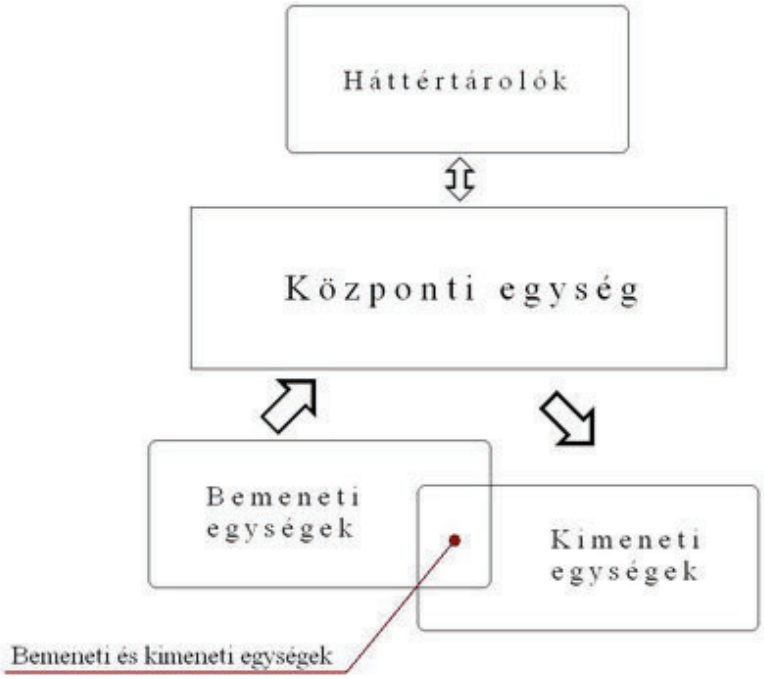
- Az adatfeldolgozó rendszert a hardver, a firmware, az operációs rendszer, a felhasználói programok (alkalmazások), a rendszerkonfigurációs adatok és a felhasználói adatok kombinációjaként mutatja be.
- Felismeri a rendszerprogramokat, és Példákat sorol fel rendszerprogramokra.
- Felismeri az alkalmazásokat, és Példákat sorol fel alkalmazásokra.

1.1.1. Alapfogalmak

Az adatfeldolgozó rendszer hardverek, hardverrel egybeépített szoftverek (firmware), operációs rendszerek, alkalmazói szoftverek, rendszerkonfigurációs adatok és felhasználói adatok kombinációjából épül fel. Az adatfeldolgozó rendszer fogadja, tárolja, feldolgozza és továbbítja az adatokat, és tartalmazza az adatműveletek végrehajtásához szükséges vezérlő funkciókat.

Az adatbázist úgy tekinthetjük mint adatok és tények olyan strukturált gyűjteménye, mint például a munkavállalók, a készlet, a vevők, és a termékek, amelyeket különböző szempontok alapján fel tudunk dolgozni, és amelyekkel különböző, az üzleti igényeknek megfelelő műveletet tudunk elvégezni. A feldolgozás az adatbázisban tárolt adatokat átalakítja kimenetekké. A feldolgozás során fontos művelet az összehasonlítás, amelynek eredményétől függhet a feldolgozás következő lépése, és az, hogy milyen adatokat fogunk tárolni. A feldolgozás fontos része az adatok megjelenítési formájának előállítása, ami alapvetően meghatározza (segíti vagy gátolja) az adathalmaz értelmezhetőségét, érthetőségét.

A hardver maga a számítógépes rendszer, az egyes komponensek – a memória, a központi feldolgozó egység (CPU), a perifériák, stb. – együttese.



1.1. ábra A számítógép információáramlási vázlata

A számítógép működéséhez hardver és szoftver szükséges, a firmware e kettő közötti kapcsolatot hozza létre. A *firmware* az a szoftverutasítás, amit véglegesen bekódolnak (hardveres úton beégetnek) az adott hardvereszközbe, például egy mikrocsipen. Ezt gyakran nevezik ROM-nak, vagyis csak olvasható memóriának, amely olyan utasításokat tartalmaz, melyet a számítógép induláskor kiolvas és végrehajt, és általában az utolsó utasításként betölti az operációs rendszert.

A szoftverek két, funkciójuk szerint alapvetően különböző típust alkotnak:
- operációs rendszerek (OS - Operating Systems),
- alkalmazói szoftverek.

Az adatfeldolgozó rendszerben tárolt adatoknak is kétféle típusát különböztetjük meg:
- rendszerkonfigurációs adatok,
- felhasználó által megadott adatok.

A rendszerkonfigurációs adatokhoz tartoznak a rendszer által használt adatok, a rendszert alkotó számítógépeket, eljárásokat és eszközöket leíró paraméterek. Ugyanakkor a felhasználó által létrehozott és tárolt adatok értelemszerűen a felhasználói adatok sorába tartoznak.

1.1.2. Rendszerprogramok

Operációs rendszer

A hardvert közvetlenül működtető szoftvert operációs rendszernek (OS - Operating Sytem) nevezzük. Az operációs rendszer egy programcsomag, amely a hardvert közvetlenül működtető programoktól a felhasználóval kommunikáló programokig terjedő széles skálán mozog.

ISO nemzetközi szabványosítási szervezet definíciója szerint az operációs rendszer: „Olyan programrendszer, amely a számítógépes rendszerben a programok végrehajtását vezérli: így például ütemezi a programok végrehajtását, elosztja az erőforrásokat, biztosítja a felhasználó és a számítógépes rendszer közötti kommunikációt."

Alapvető feladatokat lát el: kezeli a perifériákat (felismeri bemenetként a billentyűzetet), tárolja és rendszerezi a fájlokat és mappákat, az adatokat továbbítja, és értelmezhető formában megjeleníti a kimeneti

egységen, amely a legtöbb esetben a monitor. Az operációs rendszer egyfajta kapcsolódási felületként működik, az alkalmazások és a hardver (a lemezmeghajtók, a perifériák, nyomtatók, hangszórók stb.) között. Az operációs rendszer teszi lehetővé, hogy az alkalmazások zavartalanul működjenek a csatlakoztatott hardver eszközök összes részletének ismerete nélkül.

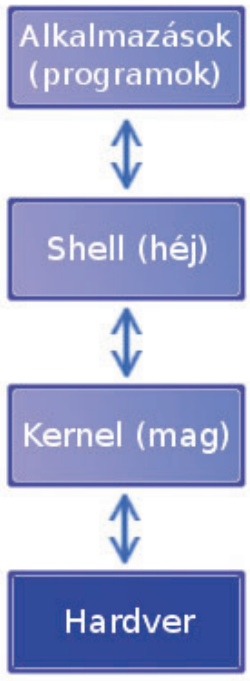
Az operációs rendszer kezeli a számítógép megosztott eszközeit, a processzort, a memóriát, a háttértárakat, a képernyőt, a billentyűzetet. Az OS feladata a számítógép hardver egységeinek vezérlése, az erőforrások összehangolása és egyszerre több, egymással párhuzamosan futó program kiszolgálása.

Összefoglalva az operációs rendszerek feladatai a következők:

- Taszkok (futó alkalmazások) ütemezése, összehangolása
- Erőforrások kezelése
- I/O műveletek megszervezése
- Hibakezelés
- A helyes és optimális működés biztosítása
- Program- és adatvédelem biztosítása
- Megszakításkezelés
- Monitoring
- Kapcsolattartás a felhasználókkal

Mivel minden program „verseng a CPU figyelméért", az operációs rendszer feladata a CPU idő kiegyensúlyozott kiosztása a programok között. Az OS-nek köszönhető, hogy nincs szükségünk a hardver működésének ismeretére ahhoz, hogy használni tudjuk a PC speciális funkcióit. Az operációs rendszer maga is egy szoftver, amely más programok futtatására egy platformot biztosít.

Az operációs rendszerek alapvetően három részre bonthatók: a felhasználói felület (a shell, amely lehet grafikus, vagy szöveges felület), az alacsony szintű segédprogramok és a kernel (mag), amely közvetlenül a hardverrel áll kapcsolatban.



1.2. ábra Operációs rendszer rétegek

Az operációs rendszereket tulajdonságaik, képességeik szerint csoportosíthatjuk, pl. a(z):

- egyszerre futtatható programok száma szerint (egy- vagy többfeladatos/multi-tasking);
- felhasználók száma szerint (egy- vagy többfelhasználós/multi user);
- kezelőfelület szerint (karakteres, grafikus).

1.3. ábra Operációs rendszerek

Az alkalmazói programok csak abban az operációs rendszerben futtathatók, amelyhez készültek. Az OS egyik legnagyobb előnye az, hogy támogatja a programok párhuzamos futtatását. A Windows felhasználó miközben egy Word dokumentumon dolgozik, zenét hallgat, játszik a gépen, ezzel egy időben fájlokat tölt le az Internetről és a háttérben dokumentumokat nyomtat. A személyi számítógépek elterjedt operációs rendszerei a Microsoft Windows 7, Microsoft Windows 8, Microsoft Windows 10, az Apple OS és a Linux. Hálózati operációs rendszerekre példaként említhetjük a Microsoft Windows Server 2008 és a Microsoft Windows Server 2012 rendszereket.

1.4. ábra Windows Server 2012 kezdőkép

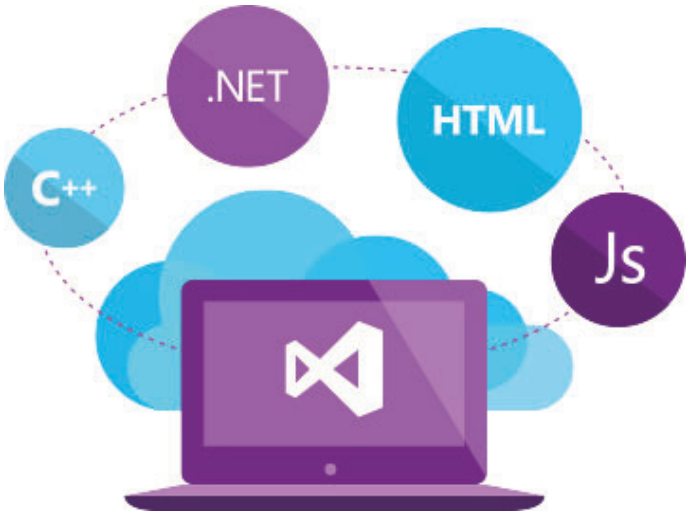
1.1.3. Alkalmazói programok

Az alkalmazói szoftver, egy adott munkaterület feladatainak végrehajtására alkalmas programcsomag.

Kifejezetten általános jellegű, valamennyire mindenki számára hasznos alkalmazások az irodai programok, programcsomagok és az üzleti alkalmazások infrastrukturális igényeit biztosító szoftverek. A speciális célú alkalmazások valamely speciális emberi tevékenység támogatására szolgálnak.

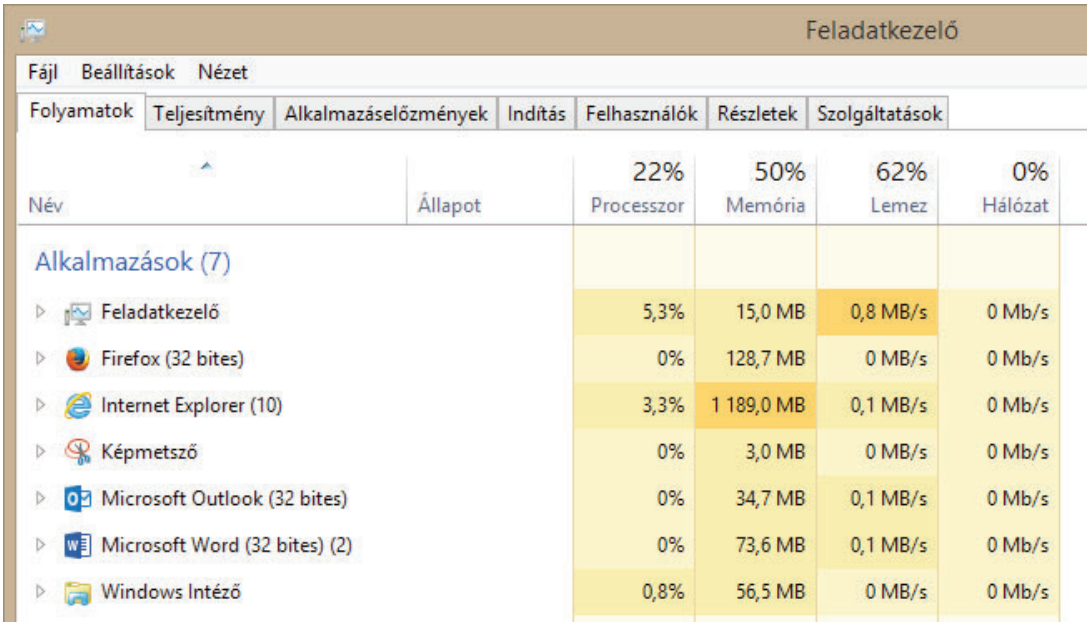
Az **irodai szoftverek** az irodai, adminisztrációs feladatok ellátásában segítenek minket. Jellemző példányaik a szövegszerkesztők, táblázatkezelők, adatbázis kezelők, levelezők, szervezők, prezentációkészítők, kiadványszerkesztők.

A másik nagy csoportja az alkalmazói programoknak az **üzleti alkalmazáso**k, melyek speciális, vagy munkavégzés céljára készülnek. Pl. számlázó programok, bérszámfejtő programok, médiaszerkesztők, a speciális feladatokat ellátó szoftverek pl. CAD programok. Ide tartoznak még a különböző rosszindulatú szoftverek (vírusok, férgek, kémprogramok) és az elhárításukra szolgáló védelmi programok (víruskeresők, reklám- és kémprogram-felderítők, titkosító programok és tűzfalak), a játékprogramok; a fejlesztési célú szoftverek, amelyek más, például az eddig felsorolt szoftverek elkészítését támogatják. (Pl.: Microsoft Visual Studio: több programozási nyelvet tartalmazó fejlesztőkörnyezet, amelyben számos programnyelv megtalálható, 2013-as verzió óta felhő alapú együttműködésre is lehetőséget nyújt.)



1.5. ábra Microsoft Visual Stúdió fejlesztőkörnyezet

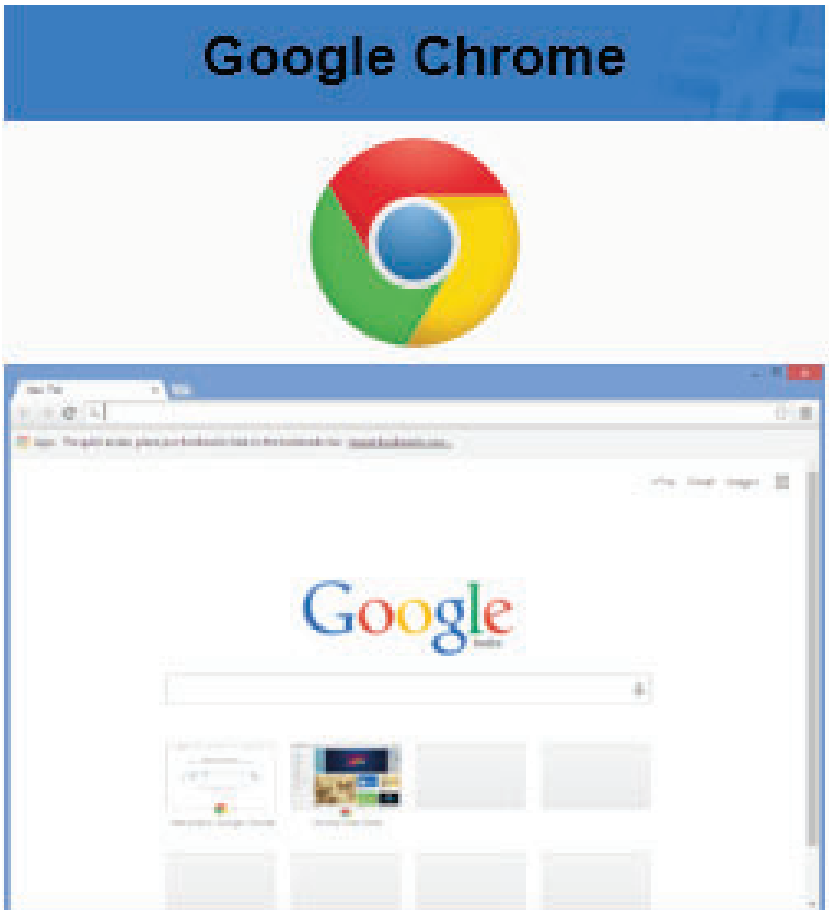
Az operációs rendszer betöltődése után a felhasználó kap egy listát a futtatható, saját feladatainak elvégzésére alkalmas programokról.



1.6. ábra Feladatkezelő ablak: Futó alkalmazások lista

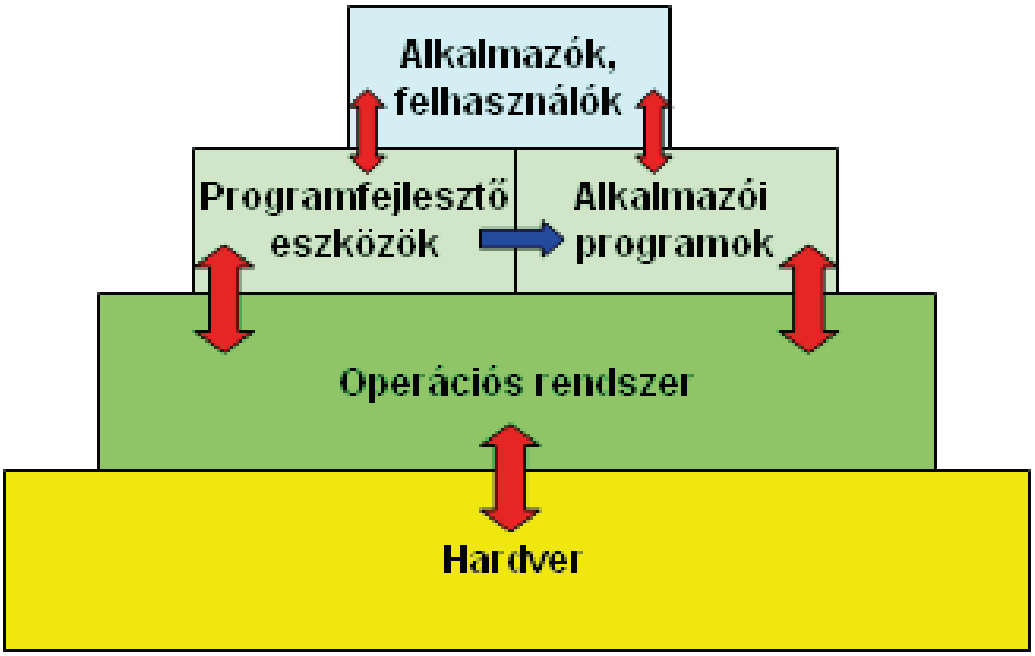
Vannak olyan, általános célú alkalmazói programok, amelyek felhasználó speciális munkaterületétől függetlenül általában minden számítógépen megtalálhatók. Ilyenek a

- böngészők (Google Chrome, Internet Explorer, Mozilla Firefox),
- az elektronikus levelező programok (Microsoft Outlook, Mozilla Thunderbird, Outlook Express, Lotus Notes),
- szövegszerkesztők (MS Word, LibreOffice, Writer, Pages)
- az adatbázis-kezelők (MS Access, MySQL, LibreOffice Base)



1.7. ábra Google Crome böngésző logo

Míg egy raktárkészlet-nyilvántartó programcsomag már kifejezetten a speciális célú alkalmazások közé sorolható, vannak olyan általános célú alkalmazások, amelyek nem tartoznak az átlagos felhasználó mindennapi munkaeszközei közé (például az adatbázis-kezelők (MS Access, MySQL) és kiadványszerkesztők (Publisher), tehát nem találhatóak meg minden számítógépen.



1.8. ábra A számítógéprendszer réteges felépítése

1.2. A rendszerfejlesztés életciklusai

Tanulási célok

- Bemutatja a rendszerfejlesztés tipikus szakaszait.
- Összehasonlítja a különböző klasszikus rendszerfejlesztési életciklus modelleket, mint a vízesés-, a spirális, a prototípus- és az inkrementális modell.
- Bemutatja egy rendszer életciklusát az elemzés, fejlesztés és bevezetés, használat és karbantartás, üzemen kívül helyezés szakaszain át.
- Felvázolja a követelményekre és a kialakításra vonatkozó előírásokat, mint a szervezeti és a műszaki specifikáció.
- Bemutatja a rendszerfejlesztés tipikus szakaszait.

1.2.1. A rendszerfejlesztés szakaszai

A számítógépes információrendszer létrehozásához fejlesztési munkára van szükség. Az információs rendszerek fejlesztésének menete ugyanazokból a lépésekből áll. Ezeket az általánosítható lépéseket a **fejlesztés életciklusának** nevezzük.

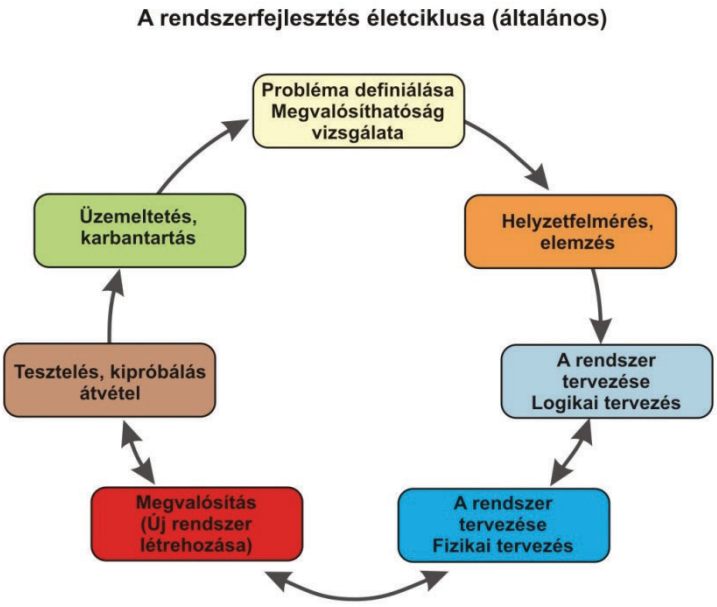
A szakirodalom többféle életutat ismer. A szakemberek egyetértenek abban, hogy van egy általános életciklus, mellyel módszertantól függetlenül, minden szervezési munka jellemezhető. A munkát lépésekben kell végrehajtani, ezek a lépések egymásra épülnek, és minden lépés bizonyos dokumentum elkészülését is eredményezi.

A rendszerfejlesztés általános életciklusát az utóbbi években a technika robbanásszerű fejlődésével technologizálták, különböző módszertanokat dolgoztak ki és olyan eszközöket is, melyekkel az egyes szakaszok tervezési, fejlesztési, dokumentálási munkáit támogatják, illetve automatizálják.

A rendszerfejlesztési életciklus az új szoftverötlet megszületésének, megtervezésének, programozásának és tesztelésének folyamata, amelynek végén a szoftver eljut a felhasználókhoz. Ideális esetben minden szakasz ellenőrzéssel zárul, és a fejlesztés csak az ellenőrzés után lép át a következő szakaszba.

A rendszerfejlesztés életszakaszai (életciklus) a következők:

- **Követelmények meghatározása** (A probléma definiálásától a felhasználói követelmények meghatározásig.)
- **Elemzés**
- **Tervezés** (Logikai és fizikai rendszer tervezése)
- **Megvalósítás, végrehajtás** (új rendszer létrehozása, fejlesztése)
- **Tesztelés**
- **Üzemeltetés, karbantartás**



1.9. ábra Életciklus modell

A követelmények meghatározása, az érintett felek (megrendelő és fejlesztő) közötti egyeztetés során kidolgozott megállapodás, az igények írásba foglalt meghatározása, amely rögzíti, hogy a jövőbeli felhasználók mit várnak a rendszertől.

Az elemzés alapja a követelmény meghatározás, eredménye a rendszerspecifikáció. A specifikáció a rendszer működésére vonatkozó igények formális ábrázolása, absztrakció, amely független a megvalósítás módjától.

A tervezés során a rendszerspecifikáció alapján elkészül a rendszer részletes terve, pontos és részletes leírással arról, hogyan fog a rendszer működni.

Megvalósítás, a rendszer kiépítése (**implementáció)** az adott tervezési dokumentumok alapján, amely igazodik ahhoz a környezethez (beleértve a fejlesztés során használt speciális hardver- és szoftvereszközöket), amelyben a rendszer működni fog. A fejlesztés lehet szakaszos, amelynek kiindulópontja egy alaprendszer, amit tesztelnek és validálnak, majd az eredmények alapján módosítanak mindaddig, amíg sor nem kerül a végleges rendszer kibocsátására.

A tesztelés összeveti a megvalósított rendszer funkcióit a tervezési dokumentumokban rögzített követelményspecifikációval. Ezt követően elkészül az átvételi jegyzőkönyv, vagy ami sokkal gyakoribb, egy hibalista a tesztelés eredményéről. Gyakran előfordul, hogy ilyenkor az eredeti elemzési és tervezési dokumentumokat is módosítani kell, és az implementációt meg kell ismételni. (A tesztelés az a munkafázis, amely az életcikluson vezérli végig az iterációt.)

A karbantartás magában foglalja:

- Követelmények módosításának kezelését
- A végrehajtási környezetek változásainak kezelését
- A hibajavításokat
- A rendszer átvitelét az új környezetbe
- A rendszer bővíthetőségét
- Adatbázis karbantartását, archiválását

Például: egy rendszer átköltözik egy önálló PC-ről egy UNIX munkaállomásra, vagy egy hálózati környezetbe. A karbantartás magában foglalja a szükséges változások elemzését, a megoldások tervezését, végrehajtását, és ennek a megoldásnak a tesztelését.

1.2.2. A rendszerfejlesztési életciklus klasszikus modelljei

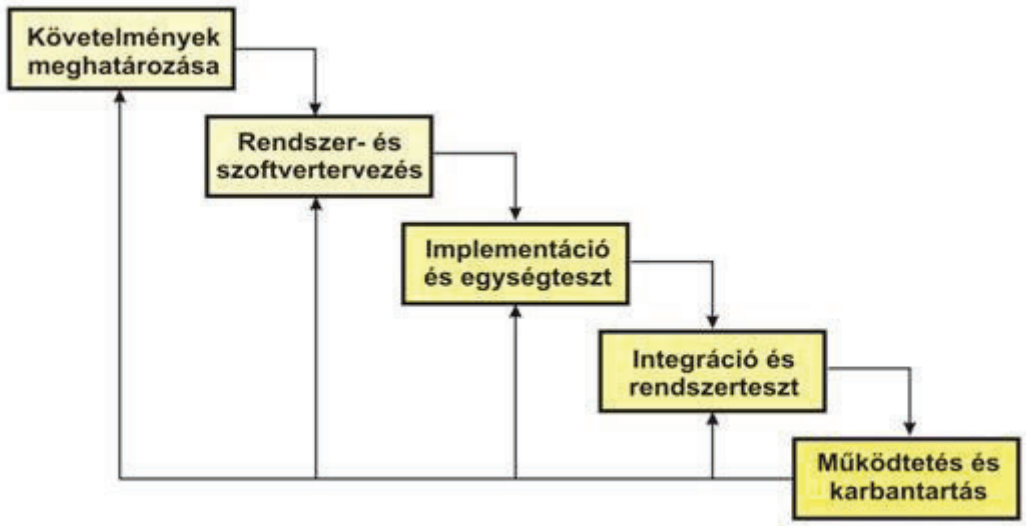
A rendszerfejlesztésnek különböző életciklus modelljei léteznek, melyek a fejlesztési módszerekkel együtt alakultak ki. Mindegyiknek megvannak az erősségeik és a gyengeségeik. Egy rendszer fejlesztési módszertana nem feltétlenül alkalmas minden projekthez. A projektek különböznek méretben, komplexitásban, költségekben és technikai nehézségekben. A választott módszer a projekt jellegétől függ.

Tekintsék át a legjellemzőbb modelleket!

VÍZESÉS MODELL

A vízesés modell megközelítést a 60-as években dolgozták ki az USA haditengerészeténél. Alapvető célja az volt, hogy lehetővé tegye a komplex katonai szoftverek biztonságos, ellenőrzött kifejlesztését. A vízesés modellben minden fázis végén a projekt csapat felülvizsgálja az adott fázist, majd lezárja azt. A fejlesztés nem folyik tovább, amíg a megrendelő nem elégedett az eredménnyel.

Ebben a lineáris módszerben minden szakasz kimenete a következő szakasz bemenetévé válik. Emiatt ez egy szekvenciális módszer, de lehetnek némi átfedések és visszahatások a fázisok között. *A szakaszok száma nincs korlátozva*, akár nyolc-kilenc lépésből is állhat. Ezek a következők lehetnek: FOGALOMRENDSZER KIALAKÍTÁSA, TERVEZÉS, KÖVETELMÉNYEK ELEMZÉSE, FEJLESZTÉS, IMPLEMENTÁCIÓ, TESZTELÉS (EGYSÉGTESZT ÉS INTEGRÁCIÓS, VAGY RENDSZERTESZT), KARBANTARTÁS. A hangsúly a tervezésben, az időbeosztáson, a megvalósítások határidejében, a költségvetéseken, és a teljes rendszerben történő egyidejű végrehajtásának a fontosságán van. Szoros ellenőrzés tartható fenn a projekt futamideje alatt, a kiterjedt írásos dokumentáció és a hivatalos felülvizsgálatok segítségével.



1.10. ábra Vízesés modell

A vízesés modell problémái:

- Lineáris, így nehéz a visszalépés a felmerülő problémák megoldására, és ez jelentősen megnöveli a javítás mind költség-, mind időigényét.
- Nem kezeli a szoftverfejlesztésben elterjedt iterációkat.
- Nincs összhangban a szoftverfejlesztés problémamegoldó természetével.
- Az integráció a teljes folyamat végén, egyben, robbanásszerűen történik. A korábban fel nem fedezett hibák ilyenkor hirtelen, együttesen jelennek meg, így felderítésük és javításuk egyaránt nehezebb feladat.
- A megrendelő csak a folyamat végén láthatja a rendszert, menet közben nincs lehetősége véleményezni azt.
- A minőség szintén csak a folyamat utolsó fázisában mérhető.
- Minden egyes fázis az előző fázis teljes befejezésére épít, ezzel jelentősen megnő a kockázat.

- A fejlesztés során a követelmények nem módosíthatók, hiszen már az életciklus elején befagyasztjuk őket.
- Már a fejlesztés kezdetén ismernünk kell valamennyi követelményt, azok későbbi módosítására vagy bővítésére nincs lehetőség.
- Elképzelhető, hogy bár a végtermék megfelel valamennyi specifikációnak, mégsem működik (pl. mert az elvárásokban vannak ellentmondások).
- Dokumentum-vezérelt, és túlzott dokumentálási követelményeket állít fel.
- Az egész szoftvertermék egy időben készül, nincs lehetőség kisebb részekre osztására.

A vízesés modell előnyei:

- Nemcsak a szoftverfejlesztésnél használható, hanem egyéb végfelhasználói projekteknél is.
- A jól érthető és kevésbé komplex projektek esetén szabályozottan rögzíti a komplexitást, és jól megbirkózik azzal.
- Könnyű megérteni.
- Egyfázisú fejlesztéseknél egyszerű a használata.
- Strukturáltságot biztosít még a kevésbé képzett fejlesztők számára is.
- Biztosítja a követelmények rögzítését, és azok nem is változnak a fejlesztés során.
- Meghatározott sablont biztosít arra vonatkozóan, hogy mely módszereket használjuk az analízis, tervezés, kódolás, tesztelés és üzemeltetés során.
- Feszes ellenőrzést biztosít.
- Ha jól alkalmazzuk, a hibák már valamely korai fázisban kiderülnek, amikor javításuk még lehetséges, és kevésbé költségigényes.
- A projekt menedzser számára könnyű a tervezés és a szereplők kiválasztása.
- Ha valaki készen van az adott fázisban rá kiosztott munkával, másik projekten dolgozhat, míg a többiek is elérik a fázis lezárásához szükséges állapotot.
- A mérföldkövei könnyen érthetőek.
- Könnyű ellenőrizni a projekt aktuális állapotát.

PROTOTÍPUS MODELL

A prototípus készítés olyan fejlesztési tevékenység, amelyben létrehozzák a szoftvernek egy befejezetlen verzióját. Majd ezt a befejezetlen programot tesztelik a felhasználók, és az ő visszajelzéseiket használják a fejlesztők a teljes rendszer tervezési javaslatainak értékelésére. A prototípus egyik előnye az, hogy miután a felhasználó az egész folyamatban részt vesz, könnyebb lesz elérni, hogy elfogadja a végleges megvalósítást.

A prototípus fejlesztés lépései a következők:

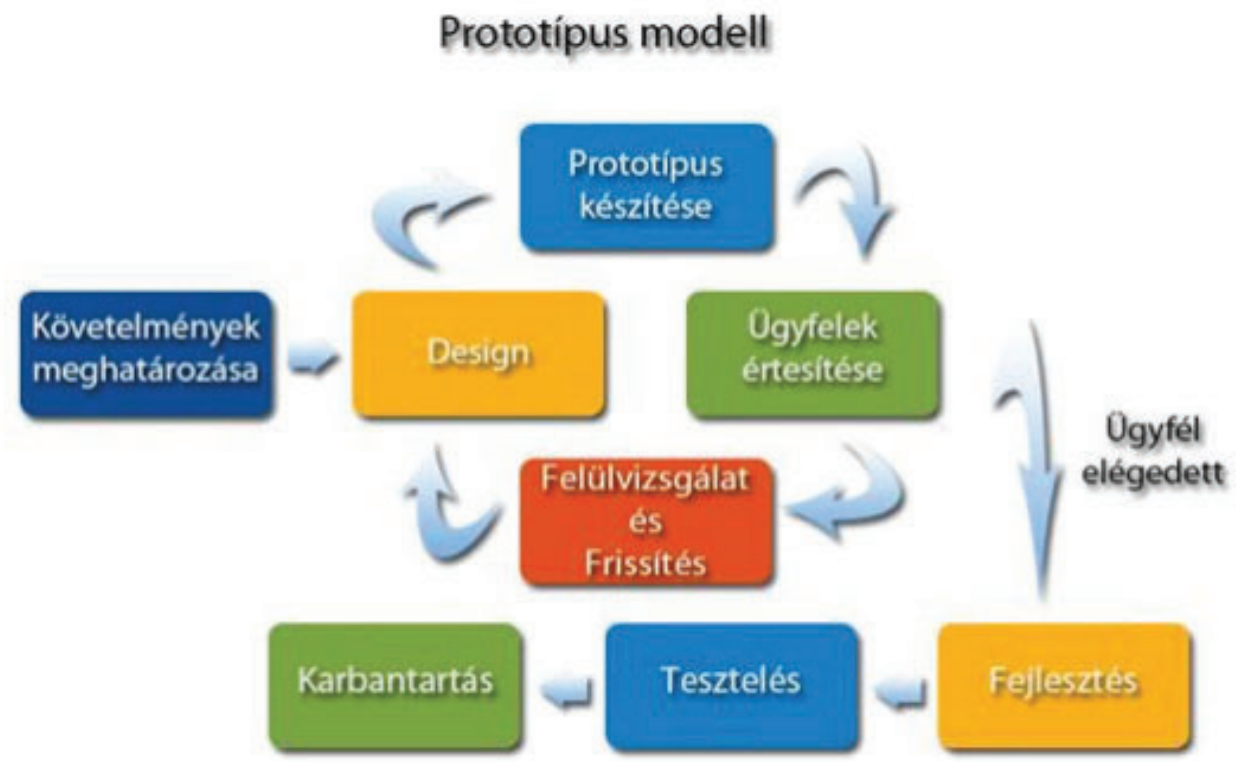
A KÖVETELMÉNYEK - beleértve az új rendszer céljait, alternatíváit és korlátait - a lehető legrészletesebb meghatározása (általában a felhasználókkal való interjúk segítségével). A követelményeket elemzik (az alternatívákat kiértékelik, a kockázatokat azonosítják és eltávolítják).

Tervezést végeznek (ennek eredménye egy ELŐZETES TERV) az elemzések alapján.

A rendszer prototípusát lekicsinyítve jön létre a TERV.

A második prototípust az első prototípus erősségeinek, gyengeségeinek és kockázatainak meghatározása után hozzák létre. Ezek használatával határozzák meg a második prototípussal szemben támasztott követelményeket.

A PROTOTÍPUS nem egy komplett módszer önmagában, hanem egy megközelítés, mely megmutatja, hogyan kezelhetők a hagyományos módszerek egyes részletei. Felosztva a projektet kisebb részletekre, megkönnyíti a fejlesztés során a változások kezelését, ezzel csökkentve a kockázatot a projektben.



1.11. ábra Prototípus modell

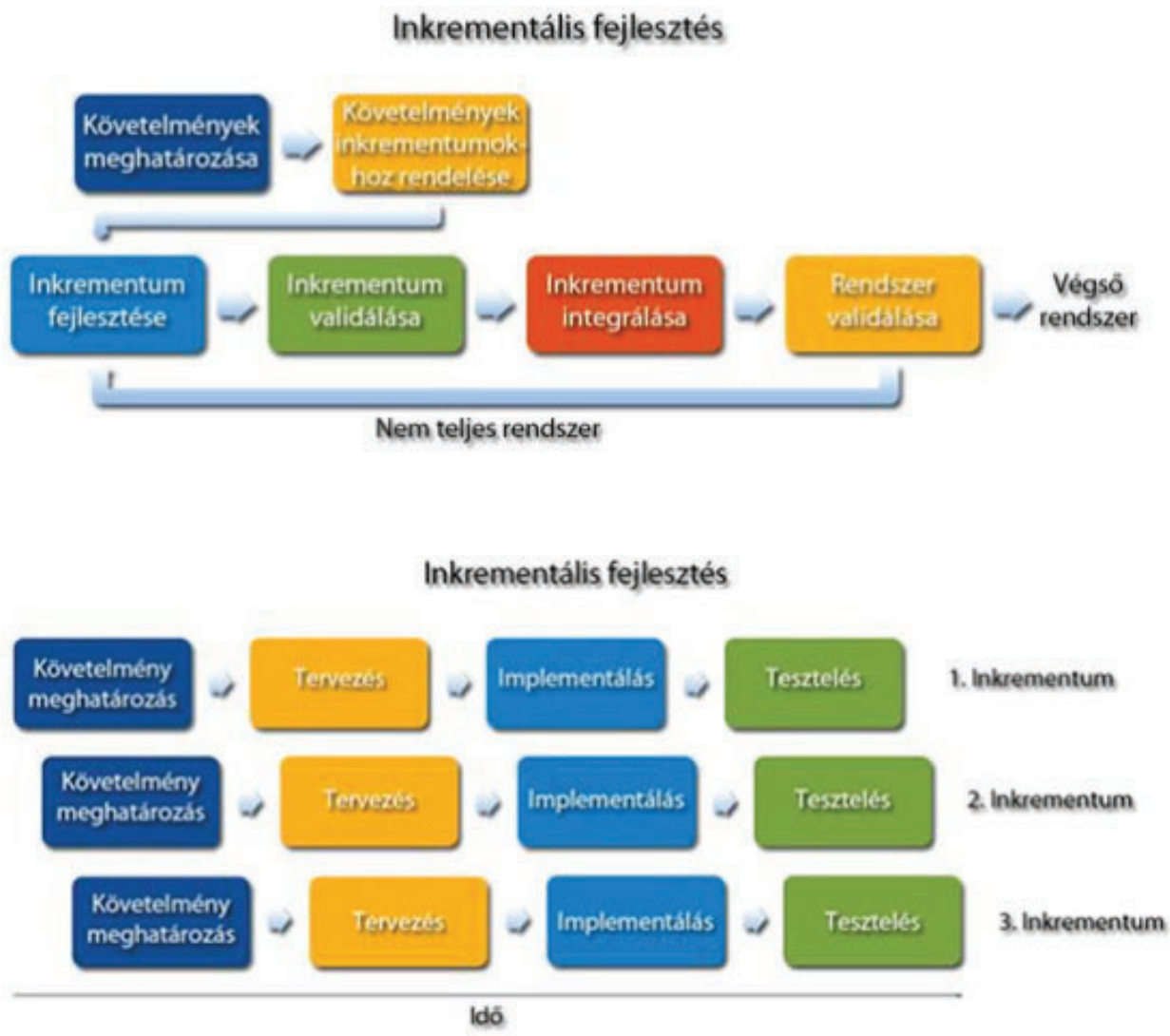
INKREMENTÁLIS MODELL

A fejlesztési módszer lényege, hogy a fejlesztési cél szempontjából lényeges, kritikus elemeket kiemelve

- szoftver-mintákat (ezek különböző verziók**, inkrementumok**) fejleszt,
- ezeket a felhasználóval jóváhagyatja,
- igény szerint javítgatja, majd
- a felhasználói megállapodás-döntés szerint tervezi meg, és
- hozza létre a végleges terméket.

Ebben - a lineáris és az ismétlődő módszer kombinációban - jelen van a számos modellre jellemző **fokozatos** megközelítés. A szegmenseket megtervezik, létrehozzák, külön-külön letesztelik, és a rendszert részenként építik fel.

A módszer előnye az, hogy a projekt korai szakaszában megszerzett tudást ki tudjuk használni a későbbi lépésekben, és ugyanakkor átláthatóbbá válik a projekt aktuális állapota az életciklus egyes pontjain. Rövidebb projektekhez nem ez a megfelelő módszer.



1.12. ábra Inkrementális modell

„V" MODELL

A szoftverfejlesztésben alapvető követelmény: a fejlesztési folyamat mindvégig **helyes, következetes** végrehajtása.

A fejlesztés egyes stádiumaihoz egymástól eltérő megvalósítási reprezentációk tartoznak. A helyes végrehajtás deklarálása azt kívánja meg, hogy bizonyítsuk ezen reprezentációk közötti egyértelmű összhang teljesülését. Ekkor azt kell bizonyítanunk, hogy a végeredményként kiadódó szoftver-termék 100 %-ig megfelel a kiindulási specifikációnak.

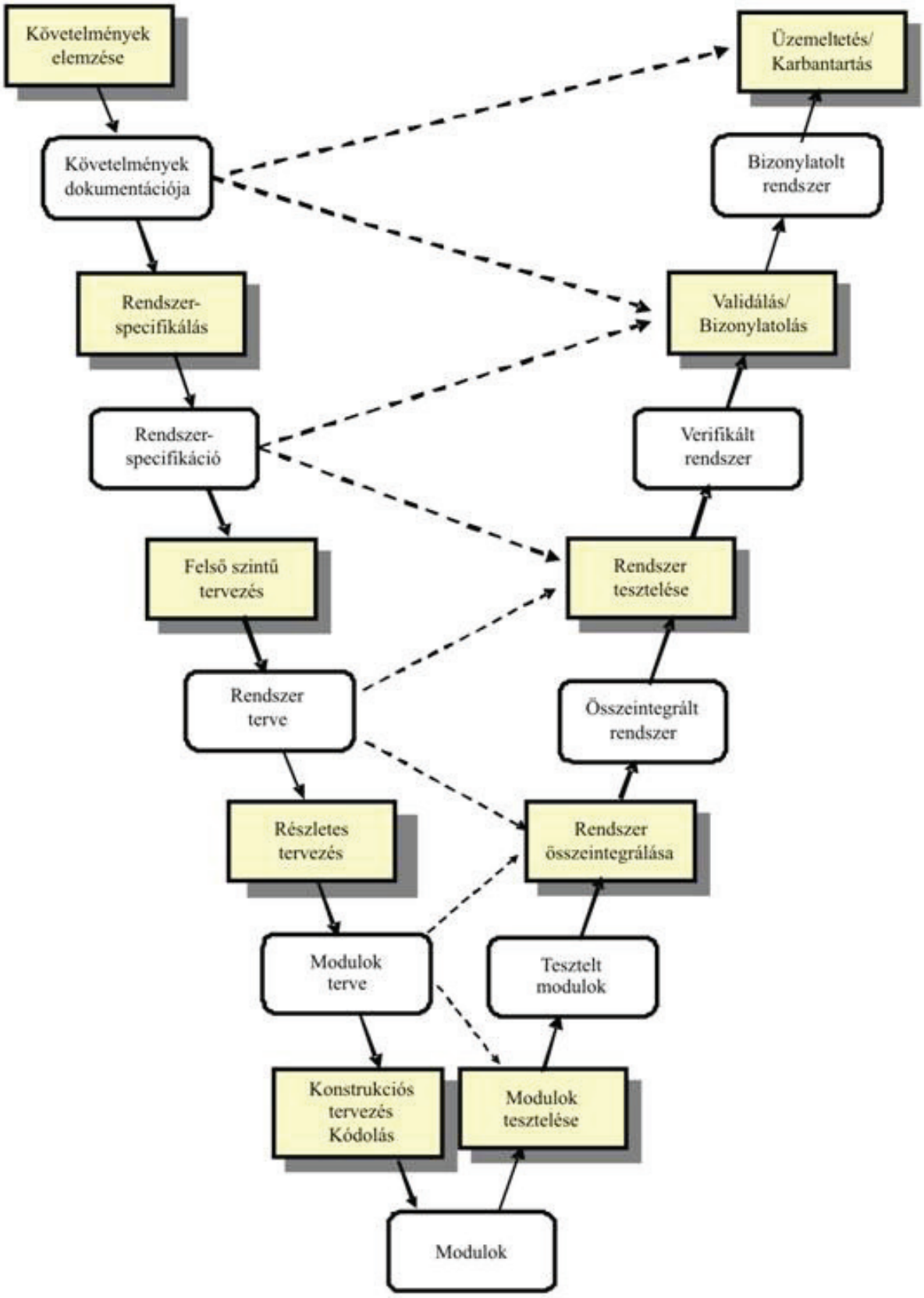
A bizonyítási folyamat elvégzése úgy logikus, hogy az egymást követő fejlesztési fázisok közötti összhang, ekvivalencia igazolását végezzük el lépésenként. Ha eltérés adódik, akkor az éppen vizsgált fázist addig kell módosítani, amíg összhangba nem kerül az őt megelőző fázissal.

Az egymást követő fejlesztési fázisok közötti összhang, ekvivalencia ellenőrzési folyamatát nevezzük verifikációnak.

Egy rendszer fejlesztési életciklusának állomásai a V-modellben:

- **Követelmények specifikálása:** Minden fejlesztési folyamat kiindulási pontját a rendszerre vonatkozó követelmények képezik. Ennek megjelenési formája: **Funkcionális követelmények** dokumentációja. A dokumentáció azt írja le, hogy milyen funkciókat és milyen módon kell a rendszernek teljesíteni.
- **Hazárdok és kockázatok elemzése:** Célja: a lehetséges veszélyhelyzetek meghatározása a rendszerben, a megelőző kiszűrés érdekében.
- **A teljes rendszer-specifikáció:** A funkcionális követelmények valamint a biztonsági követelmények együttese alkotja. Mindezen specifikációk alapján kezdhető meg a teljes rendszer konkrét tervezési folyamata.
- **Architekturális tervezés:** A teljes informatikai rendszer architektúrája hardverből és szoftverből tevődik össze. A tervezésnek ebben a fázisában azt kell eldönteni, hogy mely funkciók legyenek megvalósítva hardver, és melyek szoftver által. A választási szempontok: teljesítmény, sebesség, biztonság, költségesség.
- **A szoftver modulokra bontása:** Célja a tervezési folyamat egyszerűsítése, áttekinthetőbbé tétele, a fejlesztési feladatok részekre való szétosztása. A tervezés eredményeként a szoftver modulok specifikációja, valamint a köztük levő interfészek, ill. kapcsolódási folyamatok terve készül el.
- **A modulok elkészítése és tesztelése:** Ebben a szakaszban történik meg az egyes modulok forrásnyelvi kódjának megírása, a modulok egyenkénti lefordítása, a nyomkövetéses fejlesztői "belövése", ezt követően pedig az elkészült modulok önálló tesztelése. A **tesztelési folyamatok** előzetesen megtervezendők. A tesztelés már integráns része a szoftver verifikációs folyamatának, amelyben azt döntjük el, hogy egy-egy modul megfelel-e a specifikációjának. Ehhez az inputot a modulok terve szolgáltatja.
- **A szoftver-modulok összeintegrálása:** Miután mindegyik modul átment a tesztelésen, a teljes rendszer összeintegrálására kerül sor. Ennek a folyamatnak a végrehajtásához a bemenő információt a **modulok terve**, és a **teljes rendszer terve** képviselik.
- **A teljes rendszer verifikálása:** Ebben a fázisban eldöntendő, hogy a teljes rendszer megfelel-e a specifikációjának, vagyis funkcionálisan teljesíti-e az összes specifikációs pontot. Az ehhez felhasználandó input: a rendszer terve, valamint a teljes rendszer-specifikáció.
- **A teljes rendszer validálása:** Az döntendő el, hogy a teljes rendszer megfelel-e a felhasználói követelményeknek. Ebbe beletartozik a biztonsági feltételek teljesítésének eldöntése is: az ún. biztonságigazolás. (Ez az előírt funkciók ellenőrzésén túli ellenőrzést követel meg.) Input: a rendszer-követelmények dokumentációja, és a teljes rendszer-specifikáció.
- **Bizonylatolás (certification):** A hatósági előírások és szabványok szerinti megfelelés eldöntése, és az erre vonatkozó bizonylatok kiállítása. Input: a rendszer-követelmények dokumentációja, és a teljes rendszer-specifikáció.
- **Üzembe helyezés, üzemeltetés, karbantartás, elavulás, üzemeltetés megszüntetése**: Input: a biztonsági követelmények dokumentációja.

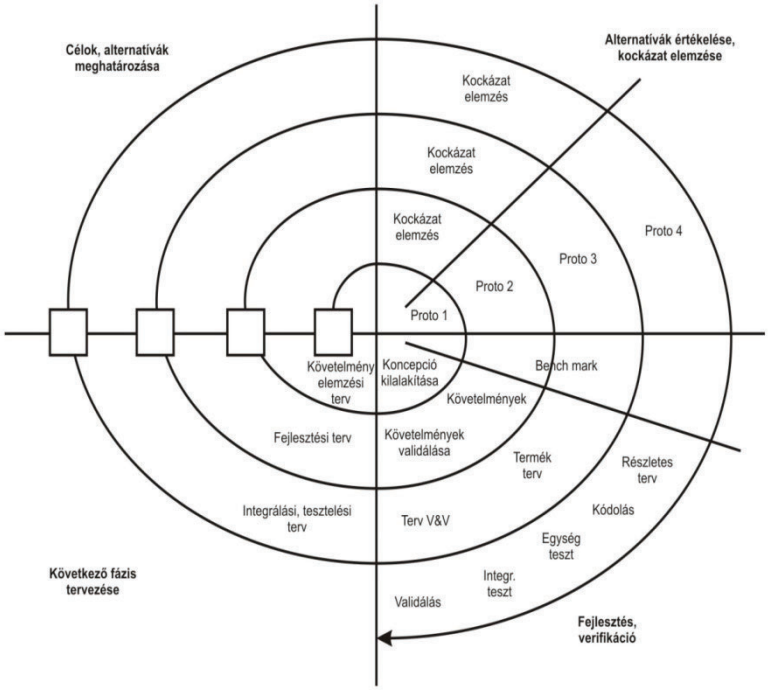
A **V-modell részletesebb megjelenítése** a következő ábrán látható. Az ábra szögletes blokkjai az egyes fázisok tevékenységét tartalmazzák, a kerekített sarkú blokkok a kimeneti eredményeket képviselik, a folyamatos nyíl az információ elsődleges áramlását, a szaggatott nyíl pedig a másodlagos információ-áramlást jelzi.



1.13. ábra „V" modell

BOEHM SPIRÁL MODELLJE

Az 1986-ban kidolgozott fejlesztési modell 4 fázis feladatainak ismétlésével és a megoldásnak minden ismétlésben egy magasabb szintre emelésével végzett fejlesztésről van szó, melyben figyelembe veszik a fejlesztés kockázati tényezőit is.



1.14. ábra Boehm Spirál modell

A spirális modell szektorai:

- Célkitűzések megállapítása
- Alternatívák értékelése, kockázatbecslés és – csökkentés
- Következő fázis tervezése
- Fejlesztés és verifikáció

Jellemzők:

- A spirál modell iterációkból áll, melyek folyamatosan ismétlődnek a projekt során.
- Valamennyi iteráció ugyanazon lépésekből áll.
- Lehetővé teszi a kockázatok korai felismerését.
- A megrendelőt minden fázisba aktívan bevonja.
- A modell elég komplex, megértése nem egyszerű.
- Jelentős kockázatkezelési szakértelem szükséges.
- A nagyszámú köztes iteráció miatt sok, végül felesleges dokumentáció születhet.

Számos más módszer van természetesen, melyek alapján a legkorszerűbb fejlesztések történnek, mely között az agilis módszerek igen elterjedtek, így többek között az ECLIPSE **EDP = Eclipse Development Process).** A fejlesztési módszer jellemzője, hogy **nyílt forráskódú, multiplatform és ingyenesen elérhető eszköz** (e tulajdonságai miatt igen népszerű).

1.2.3. A rendszer életciklusa az elemzéstől az üzemen kívül helyezésig

A tananyag korábbi fejezetében eddig a rendszerfejlesztés életciklusával foglalkoztunk, most egy más aspektusból kell a rendszerre tekintetünk. Egy rendszernek az „élete" a fejlesztés, bevezetés folyamatától kezdődően különböző karbantartásokon (módosításokon, hibajavításokon keresztül) egyszer eljut az üzemen kívül helyezésig.

Ebben a folyamatban a kidolgozásra használt program keretrendszerétől függetlenül, négy fő állomása van a rendszernek.

Egy rendszer „élete”:

FEJLESZTÉS - ÜZEMBE HELYEZÉS – MŰKÖDÉS – ÜZEMEN KÍVÜL HELYEZÉS

A rendszerek **fejlesztésére** vonatkozó tananyagot megtalálja a B 1.2.1 fejezetben.

Nézzük az üzemen kívül helyezéssel kapcsolatos legfontosabb tudnivalókat!

Üzemen kívül helyezés: A rendszer üzemen kívül helyezése arra utal, hogy amikor az új rendszer élővé válik, a régi rendszer leáll. Az előre nem látott események miatt egy ideig szükséges lehet a régi rendszerre hivatkozni. Ezen időszak után, a régi rendszert jobb leállítani.

3 féle típusú áttérés lehet:
- közvetlen átállás
- fokozatos átállás
- párhuzamos feldolgozással való átállás

A rendszer elemeit teljes körűen számba veszik. Ez magában foglalja a rendszer dokumentációját, használati útmutatókat, szállítói szerződéseket, tanagyagokat, a jelenlegi és az archivált adatbázisokat, a felhasználói hozzáféréseket és a biztonsági adatokat. Miután ez a leltár megtörtént és az összes adat archiválásra került (részben azért, hogy megfeleljenek a jogszabályoknak, másrészt pedig azért, hogy a szervezetek – szükség esetén - hozzáférjenek a korábbi adatokhoz) a rendszert üzemen kívül helyezik.

1.2.4. A szervezeti és a műszaki specifikációra vonatkozó előírások

Ahhoz, hogy el tudjunk kezdeni a fejlesztést, szükségünk van a fejlesztendő szoftver funkcióiról, működéséről szóló teljes körű leírásra. Ez a leírás az úgynevezett **szoftver követelményspecifikáció** (**SRS - Software Requirements Specification)**. A követelményspecifikáció részletesen bemutatja a felhasználó és a rendszer közötti kölcsönhatásokat, és tartalmazza azt is, hogy milyen elvárásokat fogalmaznak meg a felhasználók a rendszerteljesítményre vonatkozóan.

Az új rendszer várhatóan jobb teljesítményt nyújt, mint a régi, például abban, hogy meggyorsítja a munkafolyamatokat. Ha egy manuális rendszert cserélünk számítógépes rendszerre, akkor elvárjuk, hogy a fejlesztés növelje a termelékenységet és csökkentse a költségeket. Ez akkor is igaz, ha egy régebbi szoftvert cserélnek újra. Egyetlen szervezet sem fog a fejlesztésre pénzt fordítani anélkül, hogy az új rendszer ne járna előre meghatározott konkrét előnyökkel. A követelményspecifikáció éppen az a fejlesztési szakasz, amikor meghatározzák a várható előnyöket, amivel egyben előkészítik a későbbi munkafázist, műszaki specifikáció kidolgozását. A műszaki specifikációban nagyon részletesen és egyértelműen le kell írni a funkcionális követelményeket, hogy pontosan lássuk, milyen funkciókat kell a rendszernek elvégezni. A funkcionális követelmények leírják a rendszer „viselkedését”, a rendszer feladatait és tevékenységeit, a felhasználói igényeknek megfelelően.

Ide tartoznak bizonyos nem funkcionális követelmények is, nevezetesen a tervezésre, a minőségre illetve az implementációra vonatkozó különböző megszorítások. A nem funkcionális követelmények azok a minőségi jellemzők és feltételek, amelyek akkor keletkeznek, amikor az egyes komponensekből felépül a rendszer. Ezek a minőségi jellemzők fogják meghatározni, hogy mennyiben felel meg a rendszer az érintettek elvárásainak, azaz a ténylegesen leszállított rendszer mennyiben tesz eleget a vele szemben meghatározott követelményeknek.

A szervezeti specifikációt általában egy elemző dolgozza ki, a követelmények elemzési szakaszában. Az elemző általában az adott terület szakértője, és nem feltétlenül programozó vagy műszaki szakember.

A rendszer tervezési folyamat során számos „bemeneti” információt szolgáltatnak a rendszer működésében érdekelt személyek. A szervezeti felépítés többnyire meghatározza, hogy mely személyektől várhatóak ezek az információk. **A szervezeti specifikáció olyan kérdésekkel** fog foglalkozni, mint például, hogy **„Ki fogja használni ezt a rendszert?”, „Milyen típusú jelentéseket fognak a felhasználók igényelni?”, és „Milyen típusú képernyőn jelenjen meg a felhasználók által igényelt eredmény?”**

A fejlesztés során számos különböző technológia közül választhatunk, amelyek mindegyike hatékonyan alkalmazható. A kiválasztott fejlesztési eszköz, a választott nyelv tükrözi a meglévő fejlesztők jelenlegi tudását és szakértelmét a szervezeten belül. Ha ettől eltérő technológiát választunk, akkor egyéb problémák merülhetnek fel, mint például a szükséges szakértelem megszerzésének többletköltsége, vagy ha továbbképzésre nem kerül sor, feszültségek keletkezhetnek meglévő személyzet körében.

A műszaki specifikációt a fejlesztő csoport munkatársai dolgozzák ki, általában azokkal az elemzőkkel vagy felhasználókkal együttműködve, akik a szervezeti specifikációt készítették. Eközben olyan kérdésekre kell választ adni, hogy „**Milyen operációs rendszeren futtatható az új alkalmazás?"** Például, ha a cégen belüli szakemberek Windows platformot használnak, és döntés születik arról, hogy az új rendszer jobban futna UNIX rendszer alatt, akkor megkérdőjeleződik, hogy elegendő lesz-e a fejlesztők jelenlegi szakmai tudása. Nincs egyetlen helyes válasz, minden technológiának vannak erősségei és gyengeségei.

1.3. Szoftverfejlesztő eszközök

Tanulási célok

- Felvázolja a rendszerfejlesztés különböző szakaszaiban használatos eszközöket, mint a Lower, az Upper és az Integrált CASE (Computer Aided Software Engineering) eszközök.
- Felvázolja a különböző szoftverfejlesztések előnyeit és hátrányait.
- Felvázolja az egyszerű fejlesztő eszközök használatát a programszerkesztés, fordítás, tesztelés és hibakeresés folyamatain keresztül.

1.3.1. A rendszerfejlesztés különböző szakaszaiban használatos eszközök

A szoftverfejlesztési folyamat célja egy olyan program tervezése és fejlesztése, amely kiváló minőségű, hibamentes és jól karbantartható. A számítógéppel támogatott szoftverfejlesztés (CASE – Computer Aided Software Engineering) olyan eszközök és módszerek összessége, amelyek segítenek elérni a fent megfogalmazott eredményt, támogatják a nagyméretű információs rendszerek professzionális fejlesztését.

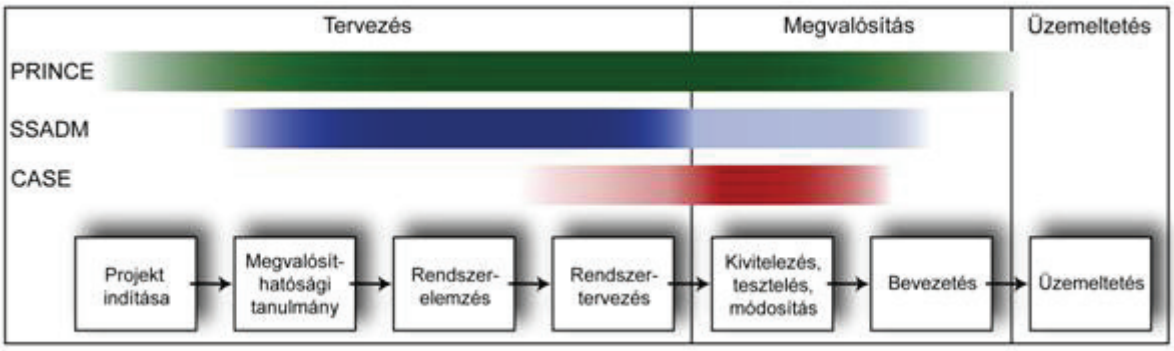
CASE eszközök:

Azokat a szoftvereket, amelyek támogatják, részben automatizálják a problémadefiniálási, elemzési, tervezési, kivitelezési feladatok elvégzését, a tesztelést és a karbantartási feladatokat, dokumentálják a fejlesztést, valamint irányítják, ellenőrzik és összehangolják a fejlesztő projekt munkáját, CASE-eszközöknek nevezzük.

Minden CASE-eszközhöz tartozik legalább egy módszertan. Vannak olyan CASE-eszközök, melyek több módszertanhoz használhatók.

Ezek az eszközök tartalmazzák azokat a technikákat, melyeket a módszertanok előírnak. Elvégzik azokat az ellenőrzéseket, melyeket a módszertanok megkövetelnek. Előállítják az adatformátumot. Elkészítik a dokumentációkat.

A teljes életciklust lefedő CASE-eszközökhöz kódgenerátor is tartozik, amely a tervezett rendszer programját létrehozza.



1.15. ábra CASE eszközök, módszertanok és az életciklus kapcsolata
forrás: Balassa Ildikó – Hegedűs Helén: Szervezési ismeretek I.-III. http://www.centroszet.hu/tananyag/szervezes2/index.html

A CASE eszközök lehetnek:

- Folyamatmenedzsment eszközök, melyeket a fejlesztési folyamatok rögzítésére és modellezésére használják.
- Projektmenedzsment eszközök, melyek segítségével tervezhetők és ütemezhetők a projektek, továbbá nyomon követhetők a folyamatok.
- Kockázatelemzési eszközök az eredmények feljegyzésére, kategorizálására és a kockázatok elemzésére.
- Követelménymenedzsment eszközök, melyek használatával rögzítik, elemzik és nyomon követik az igényeket a fejlesztés során.
- Mérési eszközök a speciális szoftver mérőszámok rögzítésére, továbbá egy átfogó minőségmérésére.
- Modellező eszközök a modellezésen belüli elemzési és tervezési tevékenységek támogatására.
- A programozási eszközök, melyeket a programfejlesztés támogatására használhatunk.
- Az Interfész-tervező eszközök a grafikus felhasználó felület (GUI - Graphical User Interface) kidolgozásának támogatására.

CASE-eszközök csoportosítása

A CASE-eszközök különböző módon csoportosíthatók. A leggyakrabban aszerint különböztetjük meg őket, hogy a fejlesztés mely fázisát hogyan támogatják, vagy milyen más funkciókat látnak el.

A leggyakrabban alkalmazott csoportosítások:

Integráltság szerint:

- **CASE Tools**, melyek legalább egy fázist támogatnak,
- **CASE Toolkits,** melyek néhány fejlesztési fázist támogatnak,
- **CASE Workbench**, mely egy egész életciklusbeli fejlesztési folyamatot támogat,
- **I-CASE Integrated Workbench,** az előzőknél komplexebb, hatékonyabb eszközök. Ezt az eszközt jellemzi az is, hogy többféle módszertan támogatásához fejlesztették ki.

A fejlesztés életciklusában való elhelyezkedés szerint:

- **Upper CASE,** mely stratégiai tervezésre, projektvezetésre szolgál,
- **Middle CASE,** mely a rendszerelemzési és tervezési fázisok munkáját támogatja,
- **Lower CASE**, mely egyszerűbb rendszerspecifikációk készítésére szolgál.

Funkcionális szempontból:

Eszköztípus	Példák
Tervezői eszközök	PERT eszközök, becslési eszközök, táblázatkezelők
Szerkesztő eszközök	Text editor, diagramszerkesztők, szövegszerkesztők
Változtatáskezelő eszközök	Követelmény követhetőségi eszközök, változásvezérlő rendszerek
Konfigurációkezelő eszközök	Verziókezelő rendszerek, rendszerépítő eszközök
Prototípuskészítő eszközök	Nagyon magas szintű programnyelvek, felhasználói interfész generátorok
Módszertámogató eszközök	Tervszerkesztők, adatszótárak, kódgenerátorok
Nyelvi feldolgozó eszközök	Fordítók, értelmezők
Programelemző eszközök	Keresztreferencia generátorok, statikus elemzők, dinamikus elemzők
Tesztelő eszközök	Tesztadat generátorok, állomány összeállítók
Nyomkövető eszközök	Interaktív nyomkövető, belövő rendszerek
Dokumentációs eszközök	Arculattervező programok, képszerkesztők
Újratervezési eszközök	Kereszthivatkozási rendszerek, program újrastrukturáló rendszerek

Folyamatok szempontjából

Eszköztípus	Specifikáció	Tervezés	Implementáció	Verifikáció és validáció
Újratervezési eszközök			x	
Tesztelő eszközök			x	x
Nyomkövető eszközök			x	x
Programelemző eszközök			x	x
Nyelvi feldolgozó eszközök		x	x	
Módszertámogató eszközök	x	x		
Prototípuskészítő eszközök	x			x
Konfigurációkezelő eszközök		x	x	
Változtatáskezelő eszközök	x	x	x	x
Dokumentációs eszközök	x	x	x	x
Szerkesztő eszközök	x	x	x	x
Tervezői eszközök	x	x	x	x

1.3.2. A szoftverfejlesztési módszerek előnyei és hátrányai

A CASE olyan szoftverfejlesztési technológia, amely automatikus eszközökkel mérnöki fegyelmezettséget visz a rendszerfejlesztésbe, a karbantartásba és a projektmenedzsmentbe. Ezzel a technológiai újítással a programozás hangsúlya a lényegre helyeződik át: a programok kidolgozásának ideje felére, harmadára csökken, hiba nélküli kódok generálhatók, és a szükséges dokumentációk is elkészíthetők.

Információs rendszerek esetén fontos tényező a nagyobb rendszeregységek összehangolása, az optimalizálás; vagyis a rendszer átvizsgálása az egyértelműség, ellentmondás-mentesség és a teljesség szempontjából.

A CASE rendszerek bevezetésének okai:

- a felhasználó és a programozó a megoldandó feladatot nagyon különböző szemszögből látja, gondolataikat gyakran egymás számára érthetetlenül fogalmazzák meg;
- a bonyolult rendszerek készítése időigényes. Fejlesztés közben a körülmények megváltozhatnak, csakúgy, mint a fejlesztő csoport összetétele;
- a programok bonyolultsága olyan mértékű, hogy ez már a követhetőség, és az áttekinthetőség rovására ment;
- a rendszernek gyorsan, rugalmasan kell követniük az eredeti követelmények állandó változásait, különben az elavulás veszélye fenyegeti a terméket.

A CASE eszközök az alábbi feladatcsoportokat végzik el:

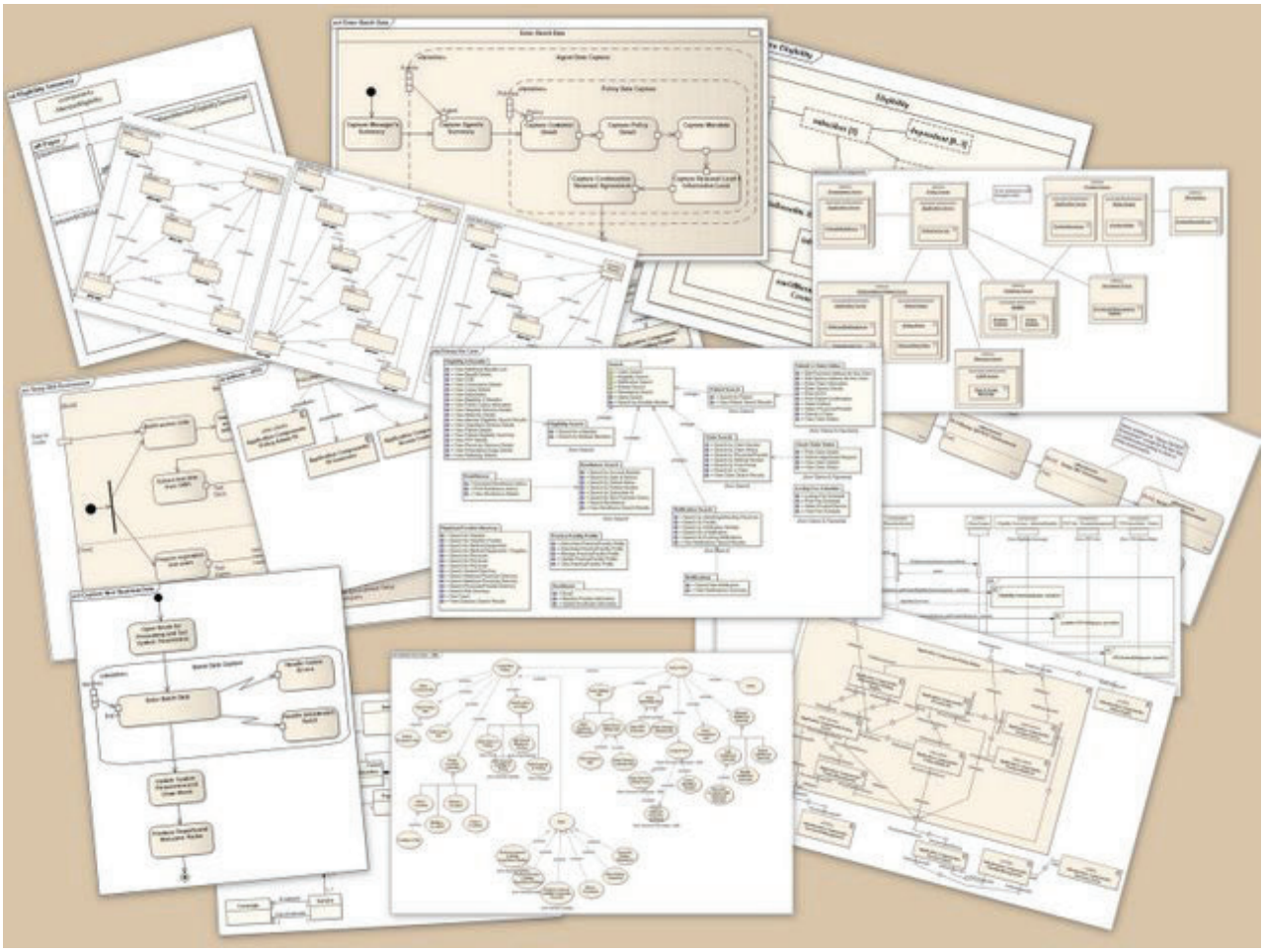
Integrálják a fejlesztési eszközöket, a fejlesztési projekt munkájában a rendelkezésre álló, lehető leghatékonyabb eszközöket alkalmazva.

Irányítják a fejlesztési folyamatot. A fejlesztési fázisok szerinti elvek világosan és egyértelműen határozzák meg azokat a feladatokat és megkívánt eredményeket, amelyeket a tervezőnek el kell végezni. A CASE eszközök követik ezeket a folyamatokat, kiválasztják a végzendő tevékenységeket, megkövetelik a végzett munka eredményét, segítséget nyújtva a feladatvégzéshez.

Irányítják a projekt tevékenységét. A projekt vezetője meghatározza az elvégzendő feladatokat, ezeket logikai és időrendi sorrendbe állítja, megadja a rendelkezésre álló erőforrásokat (emberi, gépi, idő). A CASE eszközök ezek ismeretében támogatják a projekt tevékenységének ütemezését, a projekt irányítását, kiadják, és számon kérik a munkatársaktól a feladatokat, figyelik a határidőket, nem teljesítés esetén figyelmeztetést adnak ki.

Integrálják a fejlesztési eredményeket és kezelik a fejlesztési adatbankot. A fejlesztési projekt munkatársai által készített fejlesztési eredményeket a CASE eszközök összesítik, egységesítik és ellenőrzik. Kezelik a különböző verziókat, kapcsolatot teremtenek a fejlesztők között, javíttatják a hibákat.

Összehangolják a fejlesztési fázisok munkáját. Az objektumorientált rendszerfejlesztést támogató CASE eszközök, az UML alapú fejlesztőeszközök is megjelentek. Az UML tervezőrendszerek alapvonása, hogy megfelelő grafikus eszközök segítségével hatékony segítséget nyújtanak a szoftverfejlesztési folyamat során létrehozandó modellek megalkotásában, majd a modellek alapján képesek különböző célnyelvekre automatikusan kódot generálni, ezzel az implementációt is jelentősen gyorsítják és megkönnyítik.



1.16. ábra UML diagramok

A fejlesztőkörnyezet olyan CASE eszközök gyűjteménye, amelyek támogatják a szoftver fejlesztési folyamatokat. Ezen fejlesztőkörnyezetek lehetnek:

Toolkit fejlesztőkörnyezet
Az eszköz **egy beépített termékgyűjtemény**, amely könnyen bővíthető különböző eszközök és eszközkészlet összevonásával.

Nyelv-központú fejlesztőkörnyezet
Itt maga a környezet a kifejlesztett **programozási nyelven íródott**, ezáltal lehetővé teszi, hogy a fejlesztő újra felhasználja, testre szabja és kiterjessze azt. A kód integrálása különböző nyelvekre az egyik legfontosabb kérdés a nyelvi - központú környezetben.

Integrált fejlesztőkörnyezet
Az ilyenfajta fejlesztőkörnyezetek megvalósítják a megjelenítési integrációt, az egységes, következetes és koherens eszköz és eszközkészlet felületek biztosításával.

Negyedik generációs fejlesztőkörnyezet
A negyedik generációs környezetek voltak az első **integrált környezetek**. Ezek olyan eszközök és eszközkészletek, amelyek támogatják a speciális program osztályok fejlesztését: elektronikus adatfeldolgozás és üzleti célú alkalmazások. Általában ide tartoznak a programozási eszközök, az egyszerű konfigurációs eszközök, dokumentumkezelési szolgáltatások, és néha egy kód generátor alacsonyabb szintű nyelvek előállításához.

Folyamat-központú fejlesztőkörnyezet
ELŐNYÖK (a fejlesztőkörnyezet használatának előnyei):

- A fejlesztők szemszögéből ezek a fejlesztőkörnyezetek támogatást nyújtanak a rendszer modellezéséhez, módszerek változatainak használatához.

- A technikák kiterjednek a szövegtől a matematikai diagramokig, a tervezés helyességének ellenőrzéséig, sőt még a prototípus kód előállítását is támogatják.
- Segítséget nyújtanak az írás automatizálásával, és a rendszer dokumentációjának frissítésével.
- Segíti a csapat kommunikációját, projektek és a döntések nyomon követését egyaránt.
- Támogatja a csapatmunkát és javítja a minőségét az olyan fejlesztési folyamatnak, amelyeknek van egységes módszertana.

HÁTRÁNYOK (a fejlesztőkörnyezet hátrányai közé tartoznak):

- Magasak a költségek a szervezet számára, melyet előre be kell ruházniuk technológia alkalmazásához,
- A dolgozókat (fejlesztőket) ki kell oktatnia a használatra.
- A költségek előre fizetendők, az előnyök csak hosszútávon jelennek meg.
- Ezek az eszközök csak akkor hatékonyak, ha a szabványos módszerek elfogadásra kerülnek a szervezeten belül.

1.3.3. Egyszerű fejlesztő eszközök alkalmazása

A szoftverfejlesztő eszközök lehetnek

- **teljesítmény elemzésre** szolgáló eszközök,
- **hibakeresésre**,
- **szerkesztésre**, továbbá
- **fordításra szolgáló** szoftverek.

Általában ezek az eszközök, egy készlet formájában jönnek létre, melyet „Szoftverfejlesztő készletnek (SDK = Software Developers Kitnek) nevezünk.

A szerkesztő eszközöket a kód írásához és szerkesztéséhez használják. Ha az eszközöket összehasonlítjuk, egyik előnye a másikkal szemben lehet: a sebessége, az a képesség hogy tag-eket tud adni a kódhoz, attribútumot tud adni értékeknek és szerkeszteni tudja azt. Általában a programozók nyelvi utasításokat írnak soronként, olyan nyelvekben, mint pl. a C, C++, Javascript, stb. valamilyen szerkesztő segítségével. A létrehozott fájl tartalmazza az úgynevezett forrás utasításokat, forráskódokat (source code). A programozó lefuttatja a megfelelő nyelvi fordítóval (fordítóprogram, vagy interpreter) a fájlt, melynek eredményeképpen létrejön egy szoftver.

Ez azt jelenti, hogy a **fordítóprogram egy olyan szoftver**, amely átalakítja az ember számára érthető szöveges fájlokat olyan formában, hogy azt a számítógép is megértse. A sikerhez az ember által olvasható kódnak meg kell felelnie annak a programozási nyelv szintaktikai szabályainak, amelyben íródott.

A fordító azonban csak egy program, nem tudja kijavítani az esetleges problémákat a kódban. Az ideális környezet a **rendszer tesztelése** céljából az lenne, ha lenne a rendszernek egy példánya, a rendszer pontos konfigurációjával, úgy ahogy használni fogják a működés során, és működésbe lépés után fut azon a hardver platformon, amelyet kiválasztottak, a szervezetek élő adataival.

A tesztelési eszközök célja, hogy nagy mennyiségű adatot generáljanak, hogy szimuláljanak egy bemenetet a rendszerbe nagyszámú felhasználó által, és megmérjék a gép teljesítménye során a szimulált élő alkalmazás használatát. Tesztelési eszközöket használnak annak érdekében, hogy a készített program minősége a kívánt mértékű legyen. A tesztelést el lehet végezni a kód lépésről-lépésre való végrehajtásával, és közben kiszúrni a problémákat.

Hibakereső eszközöket használnak, hogy eltávolítsák a bugokat (a **bug** a számítógépes **programhiba** elterjedt elnevezése. Előfordulásakor a számítógépes szoftver hibás eredményt ad, vagy a tervezettől eltérően viselkedik. A legtöbb bug a programozók által a forráskódban vagy a programstruktúrában vétett hibák eredménye, kisebbik részüket pedig a fordítóprogram által generált hibás kód okozza) és segítséget nyújtsanak a kód szabványoknak való megfeleléséhez.

A hibakereső egy olyan környezet, amely lehetővé teszi a kód lépésről-lépésre való végrehajtását. A hibakeresést a programban a probléma feltárásával kezdik, izolálják annak forrását, és utána kijavítják. A hibakeresést végrehajtják a rendszer **legkisebb egységén**, melynek az eredménye az **egységteszt,** aztán újra végrehajtják a tesztet mint **komponens tesztet**, amikor a rendszer részeit vetik össze, ezt követi a **rendszerteszt,** amikor a terméket a többi meglévő termékkel együtt használják, és újra tesztelnek az **ügyfél tesztelés** során, amikor a felhasználók kipróbálják a terméket egy valós helyzetben.

1.4. Rendszertesztelés és - bevezetés

Tanulási célok

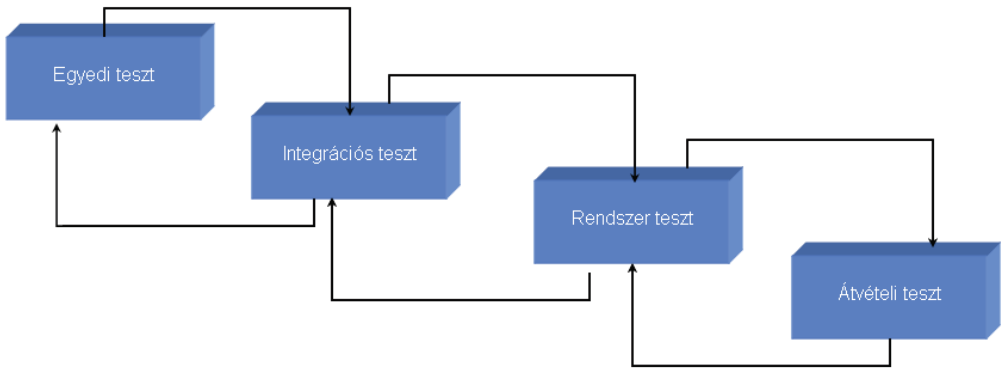
- Bemutatja a rendszerfejlesztési életciklusban alkalmazható tesztelés és felülvizsgálat különböző típusait.
- Bemutatja a rendszerimplementációs (bevezetés) fázisban felmerülő főbb feladatokat, mint a szoftver átadása a felhasználóknak, az adatmigráció, a betanítás és kezdeti támogatás.
- Felvázolja a különböző implementációs szemléletek (pl. big bang, step by step, core model, rollouts) előnyeit és hátrányait.
- Felsorolja a felhasználói kézikönyvek és technikai referenciadokumentumok tipikus részeit.

1.4.1. A tesztelés és a felülvizsgálat különböző típusai

A tesztelés, a rendszer minőségellenőrzését szolgáló folyamat. A rendszer tesztelése során a két legfontosabb ellenőrzési kritérium, a felhasználói igényeknek történő megfelelés és az üzemszerű működés. E munkafolyamat elvégzése egyrészt a fejlesztők feladata, ugyanakkor az átvételi teszt elkészítésébe a felhasználókat is be kell vonni.

A tesztelés célja a programhibák feltárása, nem annak bizonyítása, hogy a rendszer hibátlan.

A tesztelés típusai



1.17. ábra Tesztelés típusai

Az egyedi teszt valójában külön-külön ellenőrzi az egyes programfunkciók működését, ez természetesen a kezdő lépése lehet a tesztelési munkálatoknak.

A második fázisban tovább kell lépni, és azt kell leellenőrizni, hogy a programok egymással összefüggésben hogyan működnek. Ilyenkor elsősorban egy-egy menüágat veszünk nagyító alá, a csoportosítását is ellenőrizzük, ugyanakkor a közös működés vizsgálata ennek az **integrációs tesztnek** a feladata.

Az egész rendszer egységes működését, melyet már valós adatokkal célszerű elvégezni, a **rendszerteszt**

biztosítja**.** A sorban ez az utolsó olyan teszt, amelyet a fejlesztők végeznek.

Van azonban még egy fajta ellenőrzés, amely tartalmilag és az értékelés szempontjait tekintve is más, ez az **átvételi teszt**.

Az átvételi teszt a felhasználói képviselettel közösen végrehajtott feladatsor. Valamennyi teszt végezhető mintaadatokkal vagy ún. éles adatokkal. Az átvételi tesztre az a jellemző, hogy szinte kizárólag a felhasználó által biztosított konkrét adatokkal történik a rendszer működésének ellenőrzése.

Az átvételi tesztet a legmagasabb szintű tesztnek tekintjük, melynek célja annak eldöntése, hogy vajon a rendszer valóban az előre megfogalmazott felhasználói követelményeknek megfelelően működik-e?

A felhasználó által a fejlesztőtől függetlenül végzett teszteket szokták **béta teszt**nek nevezni. A béta teszt során a potenciális felhasználók egy részéhez juttatják el a programot, akik valós, éles körülmények között kezdik el használni a szoftvert, valamilyen szerződés keretében. Tapasztalataikat megosztják a fejlesztőkkel, akik ezeket felhasználják a rendszer javítására. Ha a fejlesztő végzi el ezt a tesztelést, akkor **alfa teszt**ről beszélünk.

Verifikáció[1] és validáció - V&V: (angolul: verification, validation) ellenőrző és elemző folyamatok, amelyek biztosítják, hogy a szoftver megfelel a specifikációjának és kielégíti a kliens igényeit.

Verifikáció: A szoftver megfelel-e specifikációjának? A rendszer eleget tesz-e a megadott funkcionális és nem funkcionális követelményeknek?

Validáció: A szoftver kielégíti a kliens igényeit, megfelel a vásárló elvárásainak?

Boehm megfogalmazása szerint (1979):

- Verifikáció: A terméket jól készítjük el? (Are we building the product right?)
- Validáció: A megfelelő terméket készítjük el? (Are we building the right product?)

A VERIFIKÁCIÓ hangsúlyozza a leírások betartásának fontosságát, a megrendelők kielégítésének érvényesítésével. Hogy elérjék a kívánt eredményt, mind statikus (*nem teljesítmény alapú*) és dinamikus (*végrehajtás alapú*) vizsgálati módszereket alkalmaznak.

Ebből az aspektusból kiindulva két alapvető típusa van a tesztelésnek:

1. A VÉGREHAJTÁS ALAPÚ TESZTELÉS

A végrehajtás alapú tesztelésben a tesztelés végrehajtását teszt adatok alkalmazásával végzik. Ez nem a legmegbízhatóbb formája a tesztelésnek, mert a tesztek száma korlátos minden forgatókönyvben. Ebben a típusú tesztelésben eltérő eredményeket keresünk.

Teszteljük pl., hogy:

- Hasznosság: A szoftver megfelel-e a felhasználó igényeinek?
- Megbízhatóság: Milyen gyakran hibás a szoftver? Milyen nehéz megjavítani a hibákat? Mennyire károsak a hibák?
- Erőteljesség: Milyen választ ad érvényes és érvénytelen bemenetre?
- Teljesítmény: Milyen hatással van ez a rendszer teljesítményére? Pl. memóriahasználat.
- Helyesség: A szoftver kielégíti-e az output specifikációját?

2. A NEM TELJESÍTÉS ALAPÚ TESZTELÉS

A nem teljesítés alapú tesztelés fő típusai: **a forgatókönyv**, **felülvizsgálat** és a **helyesség bizonyítása.**

Forgatókönyv/Nyomkövetés

A felülvizsgálatot egy kis csapat, beleértve a műszaki csapat képviselőit, az ügyfél, a következő folyamatot végző csapat, és a műszaki csapat vezetője végzi. Minden feltárt hibát megjelölnek, majd később a csapat

1 http://www.cid.com.ro/ksimon/svv/vv_kurzus1.pdf

egy illetékes tagja, akire ez a feladat tartozik, rögzíti azokat. A forgatókönyv bizottság nem illetékes abban, hogy ténylegesen kijavítsa a talált hibákat. A nyomkövetés folyamata legfeljebb két óra alatt lezajlik.

Felülvizsgálat

A felülvizsgálat szigorúbb eljárás, ahol a lehetséges hibákról egy ellenőrző listát dolgoznak ki és alaposan ellenőrzik, hogy jelen vannak-e, vagy sem a dokumentumban. Az ellenőrzési folyamat célja a hibák felismerése, naplózása és javítása. Ennek végrehajtásához össze kell vetni a termék dokumentációját annak forrásaival, és a termelési folyamat szabályaival. A legfőbb előnye a felülvizsgálatnak az, hogy a folyamat javítása érdekében olyan mechanizmust ad, ami létrehozza az adott, megvizsgált program dokumentumát. Így a naplózási eljárás nagyon fontos szerepet játszik annak megakadályozásában, hogy ugyanazon hibák forduljanak elő a jövőben.

Helyesség bizonyítása

A helyesség bizonyítása, egy másik formája a nem teljesítésen alapú tesztelésnek. Bizonyos körülmények között matematikai alkalmazások használhatóak annak bizonyítására, hogy a termék megfelelő-e. Azon termékekben, ahol a hibák mozgástere alacsony, mint például a biztonsági szempontból kritikus rendszerek, indokolttá tehetik a kutatási költséget az ilyen bizonyításhoz.

1.4.2. A rendszerbevezetés munkafázisa

A rendszer bevezetésének időszakában, a hangsúly a szoftverek telepítésén és a felhasználók képzésén van.

Nagyon fontos, hogy az alábbiak már rendelkezésre álljanak:

- A felhasználók a rendszer használatára vonatkozó teljeskörű ismerete
- A rendszer teljes dokumentációja álljon a felhasználók rendelkezésére
- A rendszer tesztelése és verifikációja megtörtént
- Az IT személyzet felkészültek a támogató eljárásokra
- A meglévő összes adat helyesen bekerült be az új rendszerbe.

Dokumentáció

A végső „***Felhasználói dokumentációt***" át kell adni a végfelhasználóknak. A végső „***Rendszerdokumentáció***" folyamatosan módosításra kerül, az összes frissítés és tesztelési fázisban, és átadásra kerül a támogató csoport számára.

Egy komplex rendszer működtetésének legfontosabb eszköze a szakmailag kifogástalan dokumentáció.

Az egyértelmű, használható és átfogó dokumentáció hiánya azt jelenti, hogy a szoftvert nem lehet újra implementálni, vagy fejleszteni anélkül, hogy nem végeznénk előtte egy részletes vizsgálatot, ami szintén jelentős pénzt és időt vesz igénybe. A fejlesztő csoport felelős minden rendszer-, és felhasználói dokumentációért, mely elengedhetetlen a saját munkatársak és minden felhasználó számára.

A rendszer kiépítése

A telepítés azt biztosítja, hogy az új rendszer a megfelelő szerveren és minden kliens munkaállomáson működik. Az összes szükséges adatmódosításnak vagy migrációnak teljesnek kell lennie ebben a szakaszban. A teljes ellenőrző listát használni fogják, annak biztosítására, hogy a környezet helyesen elkészüljön.

Ez a lista az alábbiakat tartalmazza:

- Minden hardver, beleértve a szervereket és munkaállomásokat, megfelelően vannak telepítve és konfigurálva.
- Minden szükséges hálózati eszközt feltelepítettek és megfelelően konfigurálva vannak.
- A működtető szoftver fel van telepítve és megfelelően van konfigurálva.
- Minden rendszer tábla fel lett töltve és ellenőrizve van, pontosak és teljes körűek.
- Minden rendszer interfész helyesen működik, úgy ahogy tervezték.

- A rendszer használatához szükséges azonosítókat és jelszavakat már kiosztották, és tesztelték, hogy megbizonyosodjanak arról, hogy azok lehetővé teszik a szükséges bejelentkezést és hozzáférést.
- Az új rendszer teljesítménye kielégítő, a tervezettnek megfelelő.
- A szabványokat és a rendszer dokumentációját már kiosztották, hogy minden felhasználó és a támogató személyzet számára is elérhető legyen.
- A felhasználói és rendszer kézikönyvek már ki lettek osztva, és rendelkezésre állnak.
- A rendszer támogató funkciói le lettek tesztelve, működnek.

Adatmigráció

Az adatok migrációja azt jelenti, hogy megtörténik a szervezetek adatátadása a régi rendszerről az újra. Ennek a folyamatnak a részeként célszerű felülvizsgálni a meglévő adatokat azért, hogy a redundáns adatokat eltávolítsuk, illetve ellenőrizzük. Ez a folyamat biztosítja, hogy csak a „tiszta" adatok kerüljenek áttelepítésre az új rendszerre.

Valószínűsíthető, hogy az új rendszer adatbázis-kezelő rendszere különböző, esetleg összeegyeztethetetlen a régivel.

Az adatmigrációs opciók az alábbiak lehetnek:
- Az adatok átvitele a régi rendszerről az újra, a régi formátumba
- A régi adatok átalakítva illetve „lefordítva" (adatok konvertálása a régi formátumról az újra)
- Új adatok létrehozása elektronikusan, vagy lehetővé teszi az új adatok rögzítését

Ha az adatokat át kell alakítani egy új formátumba, akkor időt és forrásokat kell elkülöníteni, az adat átalakítási programok írására és futtatására, és alaposan ellenőrizni kell a kimeneti eredményeket. Fontos az is, hogy használjanak egy teszt adatbázist az új rendszerben, hogy ki tudják próbálni az adat konverziós rutinokat, továbbá időt kell arra is szánni, hogy újrafuttassák őket, ha nem megfelelően funkcionálnak.

Betanítás

A képzés kulcsfontosságú tevékenység a rendszer megvalósítása közben, ugyanakkor ez bővíthető a végfelhasználói képzéssel annak érdekében, hogy ha a rendszer éles lesz, a felhasználók rendelkezzenek a szükséges tudással a sikeres működtetés érdekében.

Az elsődleges célja a végfelhasználók szoftver képzésének, a szoftverrel kapcsolatos átállásból származó gazdasági veszteség minimalizálása. Ez azt jelenti, hogy amilyen gyorsan csak lehet, kapják meg azt a szükséges képzést, ami ahhoz elengedhetetlen, hogy a munkájukat legalább olyan gyorsan és pontosan végezzék, mint a régi szoftverrel, vagy manuális módszerekkel. A következő fázisban, szeretnénk elérni azt, hogy a felhasználók, a szoftver segítségével, a munkájukat gyorsabban, pontosabban, és biztonságosabban végezzék, mint korábban.

Fontos elem a képzési terv létrehozásában az, hogy felbecsülje a technikai készségszintjét azoknak, akik ténylegesen használni fogják napi szinten a szoftvert. A következő lépésben értékelni kell a szükséges képzés módszereit. Számos tényezőt figyelembe kell venni ennek megfontolásakor:
- A felhasználói képzési szint felmérése a meghatározott igények alapján.
- A szoftver kialakításának ütemterve; valamint hogy határozatot lehessen hozni arról, hogy hogyan fog megtörténni a kiépítés, fokozatosan, vagy a teljes szervezetben egyszerre.
- Azon felhasználók száma, akiket ki kell képezni.

A szoftverszolgáltató teljes képzést fog kínálni minden felhasználó számára, de ha a felhasználók száma nagy, költséghatékonyabb módszer, ha létrehoznak úgynevezett szuper felhasználókat– vagy kiemelt felhasználókat–, akik megkapják a teljes képzést. Ezt követően viszont ők fognak teljes körű munkahelyi képzést tartani a felhasználóknak a rendszer telepítésekor.

Kezdeti támogatás

Amikor az új rendszert átadják, reálisan jelentős növekedésre számítanak az IT Helpdesk felé irányuló hívásokban, mindazon végfelhasználóktól, akiknek nehézséget okoz a rendszer használata.

Ezek egy része lehet technikai, de vannak kérdések is, amelyek nem technikaiak, és a képzés időszakában kellett volna feltenni. Gyakran előfordul, hogy nem minden felhasználó tudott részt venni a képzésen, vagy nem volt élő rendszere, amin ki tudta volna próbálni. Több különböző okból kifolyólag, bizonyos problémák nem kerülnek felszínre a bevezetésig, így a Helpdesk munkatársaknak segíteniük kell a végfelhasználót, ha probléma merül fel. Ez magában foglalja a rendszer- és a felhasználói dokumentációt, valamint a szoftverszolgáltató technikai támogatását.

Azt fogjuk tapasztalni az első használat során, hogy néhány felhasználó nem tudja ellátni a feladatait az új rendszerben, mert a hozzáférési jogaik a rendszeren belül nem megfelelőek. A helyi Helpdesk munkatársainak szükségük lesz egy olyan funkcióra, amely lehetővé teszi számukra, hogy szükség esetén megváltoztassák a felhasználók hozzáférési szintjét.

A teljes változás menedzsment rendszer nyomon követi ezeket a változásokat, amelyek életbe lépnek. A Helpdesk munkatársainak a rendelkezésére kell állnia, a kiemelt felhasználók és azok funkcionális területeinek részletes adataival, annak érdekében, hogy a közvetlen kérdésekkel a legmegfelelőbb személyt találják meg, mert néhány kérdés (mint például a „hogyan csináljam?") nem technikai jellegű.

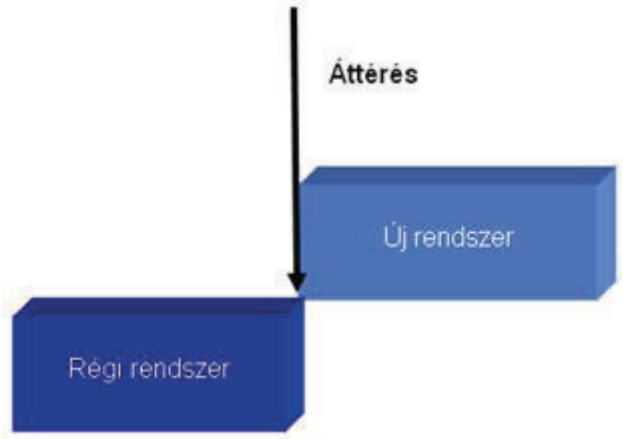
1.4.3. Rendszerbevezetési módszerek összehasonlítása

Az új rendszer bevezetése során, el kell dönteni, hogy milyen alapelveket kövessen a szervezet. Az elfogadott stratégia függ a projekt méretétől, a rendelkezésre álló időkerettől, és a szervezet méretétől.

Big Bang stratégia (közvetlen áttérés az új rendszerre)

A Big Bang stratégia során a teljes bevezetés egy fázisban történik. A stratégia előnye, hogy az intenzitás érdekében a szervezetre fókuszál, a szakaszos bevezetési megoldásnál viszonylag rövidebb idő alatt lezajlik. Ez gyakran segít abban, hogy a hosszú távú erőforrásokban várhatóan felmerülő hiányokat meghatározzuk. A módszer rövidebb időszakra sűríti az ERP projekt kellemetlenségeit és nehézségeit, de előfordul, hogy a kellemetlenségek hangsúlyosabban jelentkeznek.

A Big Bang megközelítés hátránya, hogy a projektet gyakran siettetik, figyelmen kívül hagyva a részleteket, és az üzleti folyamatok változásaihoz nem a legmegfelelőbb a szervezetek számára.

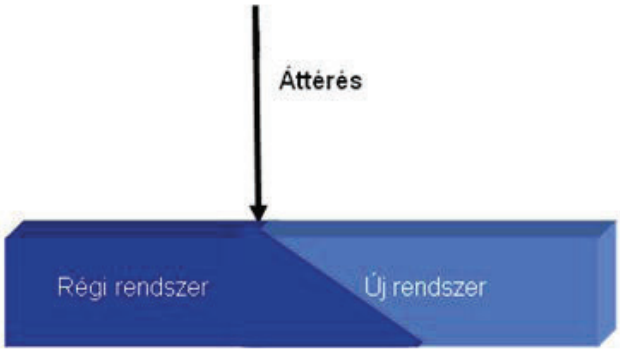


1.18. ábra Közvetlen áttérés új rendszerre

Step by step stratégia

A másik lehetőség, hogy lassabban, koncentráltabban, **lépésről lépésre történik az új rendszerre való áttérés**. Ez történhet egyrészt funkcionális üzleti terület, vagy földrajzi alapon. A vonzereje az, hogy lehetővé teszi a projekt csapatok számára, hogy idejüket a rendszer tervezésre, testreszabásra és tesztelésre fordítsák, amíg folynak a mindennapi munkák. Miután az egyes üzleti területeken már megtörtént az új rendszer áttelepítése, ugyanaz a megvalósításért felelős csapat áthozza az összes tudását az előző területről a következőre. A módszer hátránya az lehet, hogy a szakaszos projektekből gyakran hiányzik a Big Bang

projekt nyomása és összpontosítása. Ez „fásultsághoz" vezethet, amit az okoz, hogy az alkalmazottak kiégnek a folyamatos változtatások miatt. Ahelyett, hogy a projekt rövidebb ideig tartana, ezek a folyamatos változások hosszabb távon a munkavállalók elvándorlásához vezethet.



1.19. ábra Lépésről lépésre történő áttérés új rendszerre

Core Modell stratégia

Ebben a szemléletben, a szervezet ki tud fejleszteni egy Core modellt az implementáció kezdetén, majd ezt az alapmodellt megvalósítják az egész szervezeten. Előnye, hogy az alap modellt (Core Model) sokkal gyorsabban lehet fejleszteni, mint a teljes rendszert, és a korai sikerek lendületet adnak a projekt csoportoknak, ami segíti a jövőbeni fejlesztéseket.

Rollout stratégia

Több telephellyel rendelkező szervezetben a nagyméretű rendszerek teljes megvalósítása átfogó és kihívást jelentő projektek, nemcsak az IT, de a szervezeti igények szempontjából is. Ilyen esetben a Rollout szemléletnek számos előnye van. A végrehajtás költségét csökkenteni lehet, ha egy elszánt rollout csapatot hoznak létre, hogy aztán az összes „rolloutot" végrehajtsák.

A módszer hátránya, hogy a teljes implementációs idő évekig is eltarthat egy nagy szervezet esetében, bár az egyik előnye, hogy a problémákkal való szembesülés, és ezek orvoslása a korai szakaszban, nagyon könnyen kizárja a későbbi szakaszokban való előfordulásokat.

1.4.4. A felhasználói kézikönyvek és technikai referenciadokumentumok tipikus részei

Felhasználói kézikönyv

- Tartalmazza a rendszer hardver követelményeit.
- Tartalmazza a rendszer szoftver követelményeit.
- Áttekintést ad a felhasználó számára a rendszerről.
- Instrukciókat tartalmaz a felhasználó számára a dokumentáció használatáról.
- Tartalmazza a rendszer főbb parancsait.
- Tartalmaz egy hibaelhárítási útmutatót, amit a felhasználó követni tud, ha valami nem működik megfelelően.
- Tartalmazza az összes kézikönyvben szereplő kifejezés szószedetét.

Technikai Referenciadokumentumok

A technikai referenciadokumentumot a magasabb szintű tudással rendelkező felhasználók számára írják. Ilyenek például a fejlesztők, a technikai személyzet, akik támogatják a többi felhasználó munkáját. Ez nem egy felhasználói útmutató (kézikönyv), ez a dokumentum elsősorban azoknak kíván segíteni, akik támogatják és fenntartják a működő rendszert.

Ez a dokumentum tartalmazza a szoftver teljes műszaki leírását (a rendszer felépítését, alrendszereit, azok működését, a rendszer hardver és szoftver környezetét, konfigurációs, biztonsági és jogosultsági leírásokat).

1.5. Rendszerfelügyelet és –biztonság

Tanulási célok

- Különbséget tesz a fejlesztő, a teszt- és az éles környezet között, és érti a rendszerátadás strukturált megközelítésének fontosságát, mint a verziókezelő rendszerek és a szoftverdisztribúciós eljárások.
- Felismeri a rendszerhibákból eredő kockázatokat, és felvázolja a cégérzékeny adatok különböző szintű (pl. fizikai, ügyrendi) védelmére szolgáló intézkedéseket.
- Egy elosztott rendszerre lebontva bemutatja a mindennapi biztonsági rutineljárásokat, mint a biztonsági mentések, a hozzáférés-kezelés.

1.5.1. A fejlesztő-, a tesztelési- és futtatási környezet

A tesztelés, a rendszer minőségellenőrzését szolgáló folyamat. A rendszer tesztelése során a két legfontosabb ellenőrzési kritérium, a felhasználói igényeknek történő megfelelés és az üzemszerű működés.

E munkafolyamat elvégzése egyrészt a fejlesztők feladata, ugyanakkor az átvételi teszt elkészítésébe a felhasználókat is be kell vonni.

A tesztelés célja a programhibák (bug) feltárása, nem annak bizonyítása, hogy a rendszer hibátlan.

Amikor egy szoftver kidolgozásra kerül fontos tudni, hogy a fejlesztésre, a tesztelésre és futtatásra különböző munkakörnyezet áll rendelkezésre. Ezek azok a fizikai munkakörnyezetek, ahol az adott tevékenység zajlik. A fejlesztő környezetben tervezik és fejlesztik a rendszert, megírják, szerkesztik és összeépítik a kódot.

Számos programozási nyelvben létezik nyomkövetés (debugging, debugger), így könnyebb a hibák megtalálása és kijavítása. Lehetőség van lépésről lépésre végigkövetni, hol található hiba a program sorai között.

Ha mindegyik verzió kész, továbbadják a tesztelő csapatnak, akik betöltik azokat a tesztkörnyezetbe. Itt a program minden részletét átfogó tesztelést hajtanak végre, és elkészítik a tesztelési dokumentációt.

Amikor befejeződött a program tesztelése, feltöltik az éles futtatási környezetbe, ahol a végfelhasználók használni fogják.

Ez a folyamat biztosítja, hogy minden program tökéletesen legyen tesztelve és a hibákat kijavítsák, mielőtt sor kerül a futtatásra az éles környezetben. Ez csökkenti a veszélyt, hogy a felhasználók a rendszerhibák miatti leállásból eredő problémákkal szembesüljenek.

A szoftver szállítók rendszeresen jelentést készítenek az ideiglenes szoftververziókról, hogy a felmerülő problémákat megoldják a telepítés során, és felszámolják a vizsgálat során talált hibákat (bugokat). Az átmeneti szoftverkiadásokat „patch”-nek nevezik, és különösen akkor van ebből sok, ha egy új modult nemrég vezettek be, vagy ha a rendszer éretlen, és nem olyan erőteljes, mint egy tökéletesen kidolgozott termék. A beszállítók gyakran megkérik az ügyfeleket a patch-ek telepítésére nem sokkal a megjelenésük után, ami nagyon pontos verziókövetést követel, és nem mindig egyszerű és olcsó feladat, ráadásul a patch-ek nem feltétlenül fedik le azt szolgáltatást részét, amire a szervezetnek szüksége van.

A szolgáltatók a további új verziókat rendszeresen megküldik ugyanazon szerződés mellett. Ezek a verziók az úgynevezett „upgrade”-ek, amik tartalmazzák bővítések javítására mind a hibajavításokat (bug fixes), mind a rendszerhez adott új funkciókat.

Miközben az upgrade-ek (frissítések) biztosítják a naprakészséget, a továbbfejlesztett és erőteljesebb szoftvert, ugyanúgy azt is jelentik, hogy ezek nagyon költségesek, mert megkövetelik a gondos telepítést, az alapos tesztelést, a részletes vizsgálatot, hogy megértsük a funkcionalitás változásait és jelentős átképzésre lehet szükség, ahol a módosítások érintik az üzleti folyamatokat.

Időről-időre a beszállítók is lényegesen megújított szoftver verziókat kínálnak, amelyek jellemzően tükrözik a szoftver telepítési környezetében bekövetkezett jelentős technológiái változásokat. Így például az elmúlt években a legtöbb nagyobb szoftververzió már internet alapú rendszerben fut, amelyek tartalmaznak olyan web –alapú alkalmazásokat, mint az e-közbeszerzés, e-business és e-learning. Bár ezt általában nem szükséges frissíteni, üzleti célból mégis ott a kísértés a szoftver telepítésére, a legújabb technológia előnyeiből való részesedés reményében.

1.20. ábra E-tananyag rendszer, html-szerkesztő rendszer, e-business rendszer

1.5.2. Rendszerhibákból eredő adatvédelmi kockázatok

A szervezet éltető eleme az adat. Minden szervezet rendelkezik olyan adatokkal, amelyek mindenféle érzékeny információt tartalmaznak, beleértve az összes tevékenységet, minden ügyféladatot, termék-, ár-adatokat és még sok egyebet. A nagyméretű adathalmazt, azt a szervert, ahol ezek az adatokat tárolják, komoly védelemmel kell ellátni. Biztosítani kell

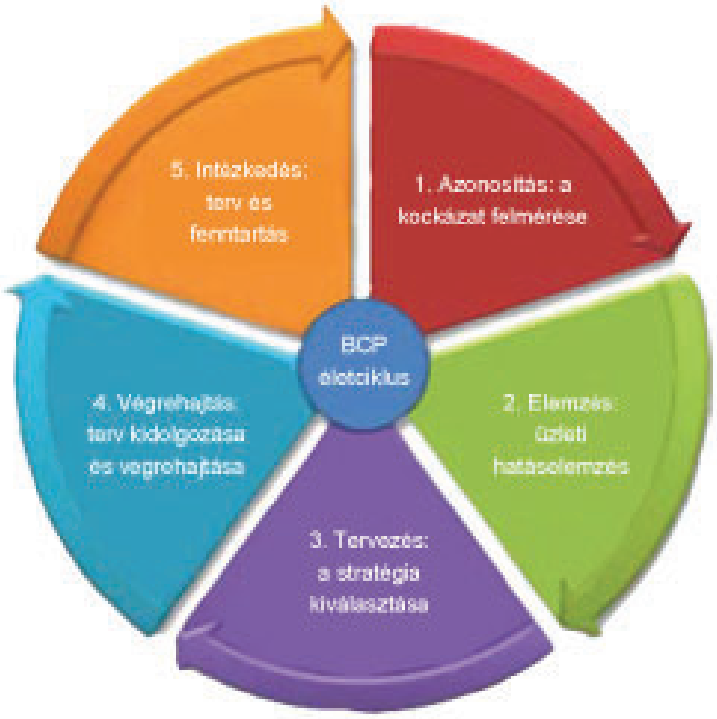
- a rendelkezésre állást: Az adatoknak mindig rendelkezésre kell állniuk az arra jogosult munkatársak számára ahhoz, hogy feldolgozhassák vagy módosíthassák az adatokat.
- a megbízhatóságot: Az adatok mindig megbízhatók legyenek. Nem lehetnek kétségek az adatok pontosságával kapcsolatban. Ha az adatok integritása bármilyen módon sérül, a rendszer használhatatlanná válik.
- a védettséget a jogosulatlan hozzáférésektől: Csak azon felhasználók, legyenek képesek az adatokhoz való hozzáféréshez, akik jogosultak erre, és a hozzáférések szintjét dokumentálni kell, és frissíteni szükség esetén.

Rendszerhiba esetén minden adatnak elérhetőnek kell lennie, és a visszaállítás után nem szabad adatvesztésnek bekövetkeznie.

Az adatvédelem egyik módja az, hogy az IT csapat megadott időközönként (általában naponta) lementi a szervereken tárolt összes felhasználói adatot. Az adatokról biztonsági mentés készül merevlemezre és egy külső helyre (off-site), ami védelmet nyújt a szerverhibákkal szemben. Ha ez megtörténik, az operációs rendszer könnyen újratölthető egy helyettesítő szerverre, és a felhasználói adatok helyreállíthatók. A külső adattárolás kritikus fontosságú, hogy az adatok ne vesszenek el, ha a helyszínen tűzeset vagy árvíz következik be. A szervezetek életében a biztonsági házirend és a mentési (katasztrófa) terv elengedhetetlen dokumentáció minden esetben, felhő alapú szolgáltatás esetén is.

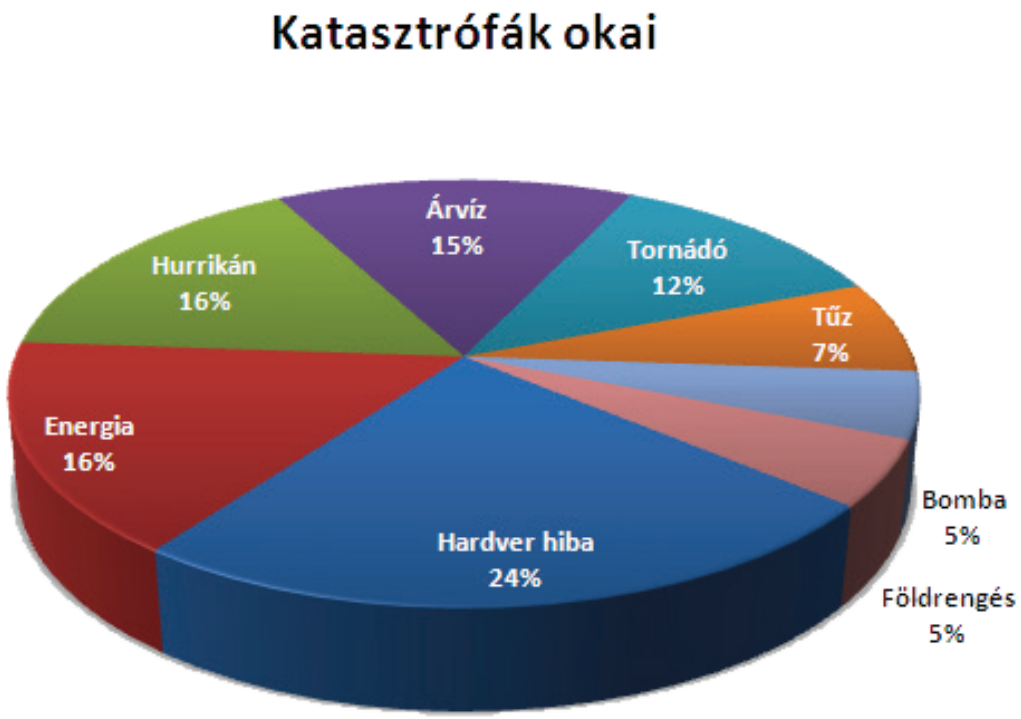
Fontos követelmény a mentések a számítógépektől távol más helyen való tárolása, annak érdekében, hogy az adatvesztést akkor is elkerüljük, ha a telephelyet érintő elemi csapás tönkre tenné a számítógépeket.

Sok szervezetnek létezik üzletmenet-folytonossági terve (BCP: Business Continuity Plan) és/vagy katasztrófa-elhárítási, megsemmisülés utáni helyreállítási terve (DRP: Disaster Recovery Plan), azaz egy megállapodása a helyi hardver kereskedővel. Ez a megállapodás biztosítja, hogy a szervezet egy nála bekövetkezett katasztrófa esetén igénybe vehesse a kereskedő szolgáltatásait, amíg újraépítik a létfontosságú szervereket, és helyreállítják a legfrissebb adatverziót. Erről a helyről hajthatják végre a mindennapi műveleteket, amíg a szervezetben a helyzet nem normalizálódik.



1.21. ábra BCP életciklusa

A katasztrófavédelmi dokumentum tartalmazza az összes olyan szerver listáját, amelyeket egy katasztrofális hiba után helyre kell állítani. Felsorolja a kulcsfontosságú személyzet, a hardver kereskedő, és más fontos személyek nevét és telefonszámát. Részletesen ismerteti a helyi mentéseket, és a hozzájuk tartozó szervereket. Tartalmazza még a kritikus rendszerek rendszergazdai jelszavait is.



1.22. ábra Katasztrófák okai

Számos lépést kell megtenni annak érdekében, hogy az adatok mindig biztonságosak és védettek legyenek.

A szervezetnek rendelkeznie kell adatvédelmi szabályzattal, melynek összhangban kell lennie a hatályos jogszabályi rendelkezésekkel, ilyen a 2011. évi CXII. törvény az információs önrendelkezési jogról és az információszabadságról szóló rendelete.

Ez a dokumentum arra szolgál, hogy iránymutatást nyújtson minden alkalmazottnak az adatok tárolásáról, szállításáról, és feldolgozásáról. Ez magában foglalja mind az elektronikus adatokat, mind a papír alapú információkat. Sok szervezetben van egy „információ menedzser", aki biztosítja, hogy az adatvédelmi szabályzat minden munkavállaló számára hozzáférhető legyen, és be is tartsák azt. Az adatvédelmi szabályzatban meg kell határozni, hogy kik férhetnek hozzá az információkhoz, és hogy az adott információ milyen védettségi osztályba van besorolva.

Tartalmaznia kell egy iránymutatást azon információk tárolásához, melyek a cégen belül, illetve külső tárolón vannak elhelyezve. Meg kell határoznia, hogy mikor és hogyan kell az információkat megsemmisíteni, valamint azt, hogy az IT vezetőnek milyen feladatai vannak az adott információ elektronikus másolatának megőrzése érdekében.

A szervereken és a számítógépeken tárolt adatokat is védeni kell. Az adatvédelmi irányelv határozza meg, hogy ki milyen adathoz fér hozzá, és részletesen kitér azokra az eljárásokra, amelyeket be kell tartania az informatikai osztálynak, amikor kérés érkezik egy bizalmas adathoz való hozzáféréssel kapcsolatban.

1.5.3. Mindennapi biztonsági járások

Az informatikai hálózatok alkalmazásával egyszerűbbé váltak mindennapjaink, könnyebben elérhetők az adatok. Kétfajta biztonsági problémát kell szem előtt tartani: a hálózati infrastruktúra biztonsága és az információbiztonság.

Előbbi a hálózati kapcsolatot biztosító eszközök fizikai biztonságáért, és az illetéktelen hozzáférések megakadályozásáért felelős. Az utóbbi azt jelenti, hogy védeni kell a hálózatban áramló adatcsomagokat.

Fontos, hogy megakadályozzuk az információk illetéktelenül történő nyilvánosságra hozatalát, adatmódosítását, valamint az információk, adatok eltulajdonítását. A leggyakoribb külső hálózati fenyegetések a következők:

- vírusok, férgek és trójai falovak
- kémprogramok és reklámprogramok (spyware és adware)
- hacker támadások
- adatlehallgatás és a lopás

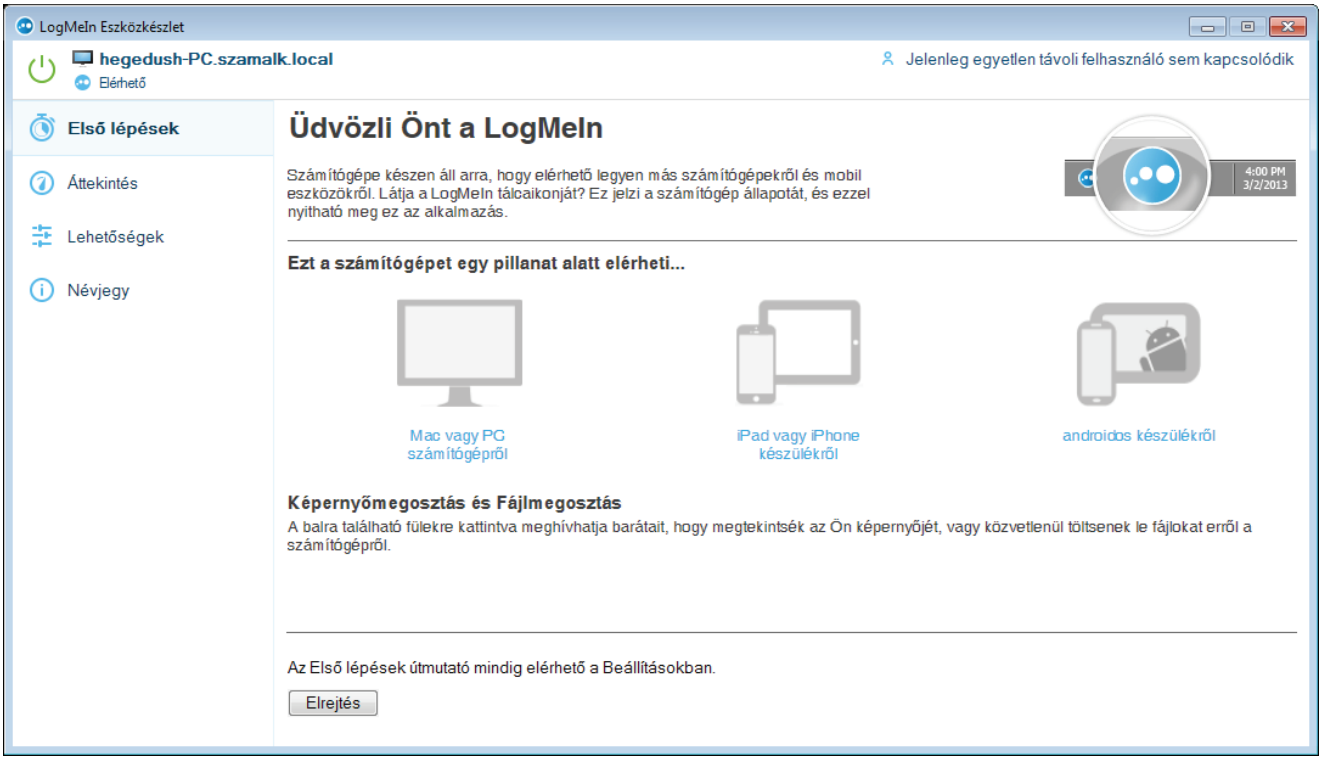
Hogyan lehet a fenyegetettségek ellen védekezni?

- Vírus- és kémprogram védelemmel
- Tűzfal szűréssel (szoftveres tűzfal, hardveres tűzfal)

Hozzáférés-kezelés

Az információ biztonság felelőssége a szervezetnél az IT osztály hatásköréhez tartozik, az ő felelősségük, hogy megfelelő eljárások létezzenek a szervezet adatainak megvédésére. Először is a fizikai környezetet kell biztosítani, hogy csak azok az ember férjenek hozzá, akik a személyzetnek azon tagjaik, akik kezelik ezeket a rendszereket.

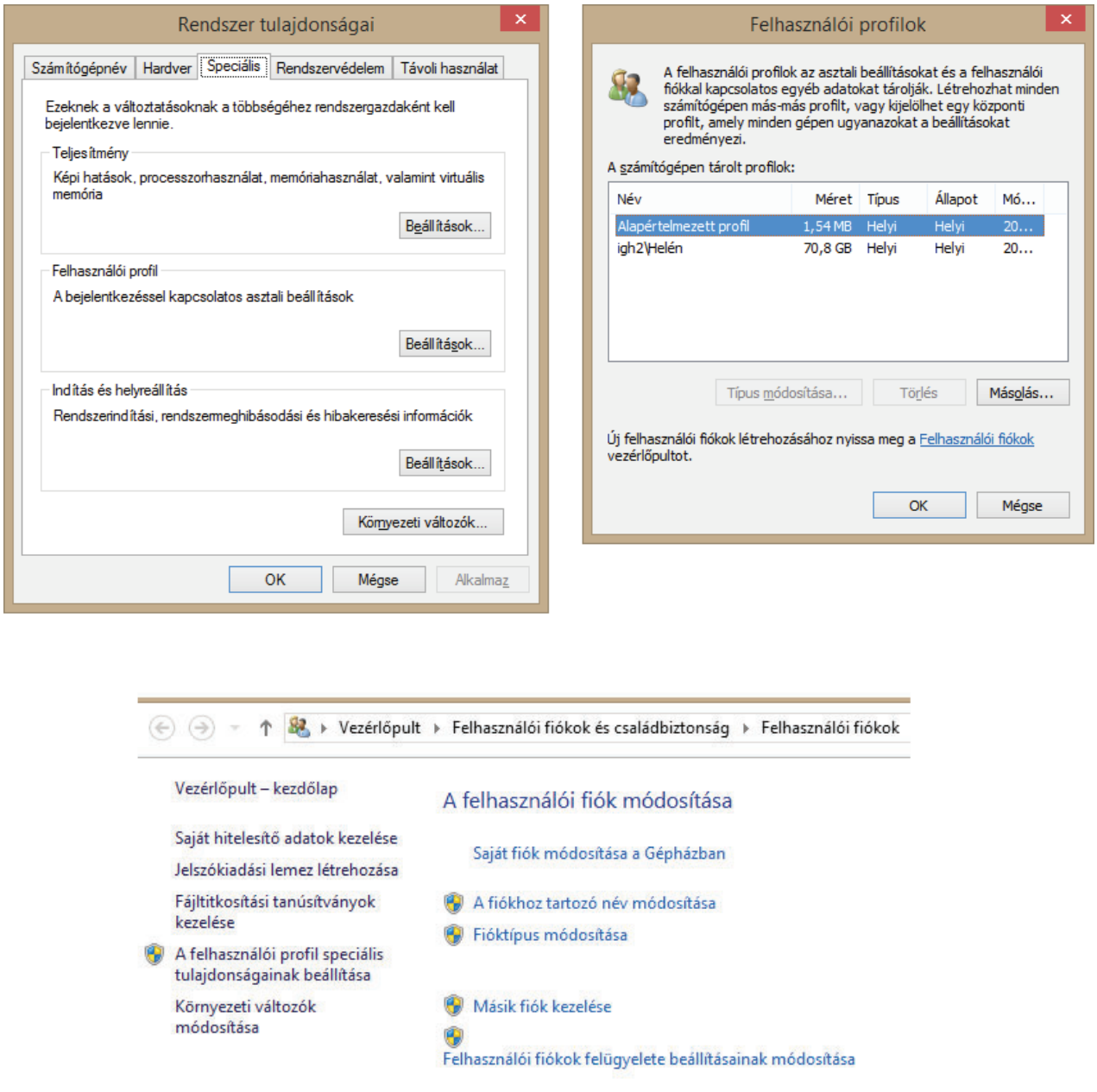
A távoli hozzáférést (ilyen pl. LogMein) a szerverekhez minimálisra kell csökkenteni, és csak az arra jogosult személyek számára elérhetővé tenni. Azoknak a látogatóknak, akik távoli hozzáférést igényelnek a szerverhez, biztosítani kell, hogy csak ellenőrzött hozzáférés mellett, és csak akkor férhessen hozzá a rendszerhez, ha egy erre felhatalmazott alkalmazott meghívta őt a rendszerbe. Nem jó gyakorlat hozzáférést adni a látogatóknak a termelési rendszerekhez, bármilyen karbantartást, amit nekik kell elvégezni, a fejlesztő rendszeren javasolt megvalósítani.



1.23. ábra Távoli asztal kapcsolat LogMein programmal

Léteznie kell egy olyan eljárásnak, amely részletesen leírja azokat a folyamatokat, amely során a munkavállalók hozzáférést kérnek a fájlokhoz vagy a mappákhoz. Ha egy új alkalmazott csatlakozik a szervezethez, kell, hogy legyen egy olyan eljárás, amely elmagyarázza az adatokhoz való hozzáférésekhez szükséges lépéseket, a megfelelő vezetői engedély alapján. Ezt a jóváhagyást írásban kell kérni és ezeket a kérelmeket ellenőrzési célból nyilvántartásba kell venni. Azt a biztonsági szolgáltatást is fel kell használni, amikor a munkavállalónak a megfelelő rendszerhez való hozzáférését vizsgálják. Minden munkavállalót kötelezni kell arra, hogy a számítógépekre egy egyedi azonosítóval és jelszóval lépjenek be, valamint az egyes dol-

gozóknak hozzanak létre egy felhasználói profilt mindegyik rendszeren. Kell egy olyan eljárás, amely részletesen megszervezi a felhasználói jelszó megadásának követelményeit, beleértve a jelszó összetettségét, és a megváltoztatás gyakoriságát is.



1.24. ábra Felhasználói profil beállításai Windows 8 alatt

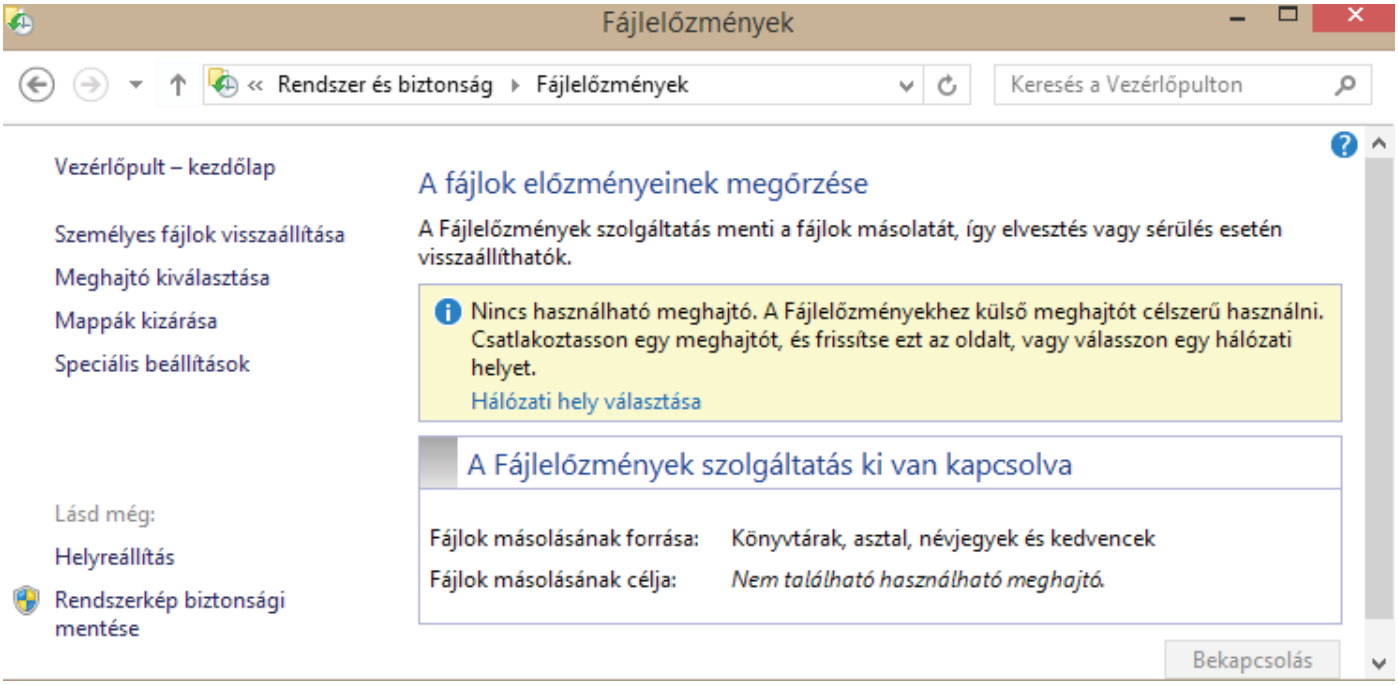
A rendszeres biztonsági ellenőrzést egy külső audit csapat által kell elvégezni minden szerveren. Ezt az ellenőrzési eljárást úgy kell kialakítani, hogy ellenőrizve legyen a szervezetek biztonsági és információs irányelveinek betartása.

Biztonsági mentések

A biztonsági mentéseknek az a célja, hogy a szervezet adatainak rendelkezésre állását biztosítsák. A biztonsági mentés folyamata magában foglalja a szerveren lévő adatfájlok és mappák másolását, és áthelyezését egy olyan adathordozóra, amit fizikailag el tudnak távolítani a szerver szobából, és biztonságos helyen tudják tárolni, hogy szükség esetén visszakereshető legyen. Ez az adathordozó általában egy több gigabájtnyi adat tárolására alkalmas, nagy adatsűrűségű szalag.

A biztonsági mentések gyakorisága nagymértékben függ az adatoktól és attól, hogy ez az adat mennyire létfontosságú a szervezet számára. Azokat az adatokat, melyek

ritkábban változnak és nem létfontosságúak, ritkábban kell menteni, mint azokat az adatokat, amelyek naponta változnak és létfontosságúak a szervezet számára.



1.25. ábra Rendszer biztonsági másolata

A napi adatmentést minden létfontosságú szerveren el kell végezni, és egy biztonságos helyen tárolni, ahol könnyen elérhető, és lehívható a rendszer meghibásodása esetén. Ez a hely a teljes biztonság érdekében egy külső hely (off-site) legyen. Sok szervezetnek vannak biztonsági mentései a helyi bank páncélszekrényében, és van a bankkal egy megállapodásuk, hogy nyitvatartási időn kívül hogyan férhetnek hozzá ezekhez. Számos kiváló szoftvercsomag van, ami segíti a biztonsági mentés folyamatát. Egy probléma, amivel a szervezetek találkozhatnak, hogy mivel a biztonsági másolatoknál sok adatot kell lementeni, logikusan ez sok időt vesz igénybe, így a biztonsági mentés hatással lehet a felhasználói adatokhoz való hozzáféréshez. Ezért a biztonsági mentéseket általában éjszaka végzik, amikor a felhasználóknak az adatokhoz való hozzáférése minimális.

1.6. Rendszerfejlesztési trendek

Tanulási célok

- Bemutatja a rendszerfejlesztés szokványos és innovatív megközelítéseit és szabványait, mint az ISO12207, a SEI/CMMI és az agilis módszertanok.
- Érti a jelenlegi műszaki architektúra fejlesztések hatását, mint a két-, illetve háromszintű kliens-szerver variánsok, az n-szintű web alapú, a szolgáltatásorientált architektúrák vagy az örökölt nagyszámítógép rendszerfejlesztésen alapuló kiterjesztése és integrációja.
- Bemutatja a korszerű „rendszerek rendszere" komplexitását és e komplexitás menedzselésének megközelítéseit, mint az autonóm rendszerek.

1.6.1. A rendszerfejlesztés szabványai

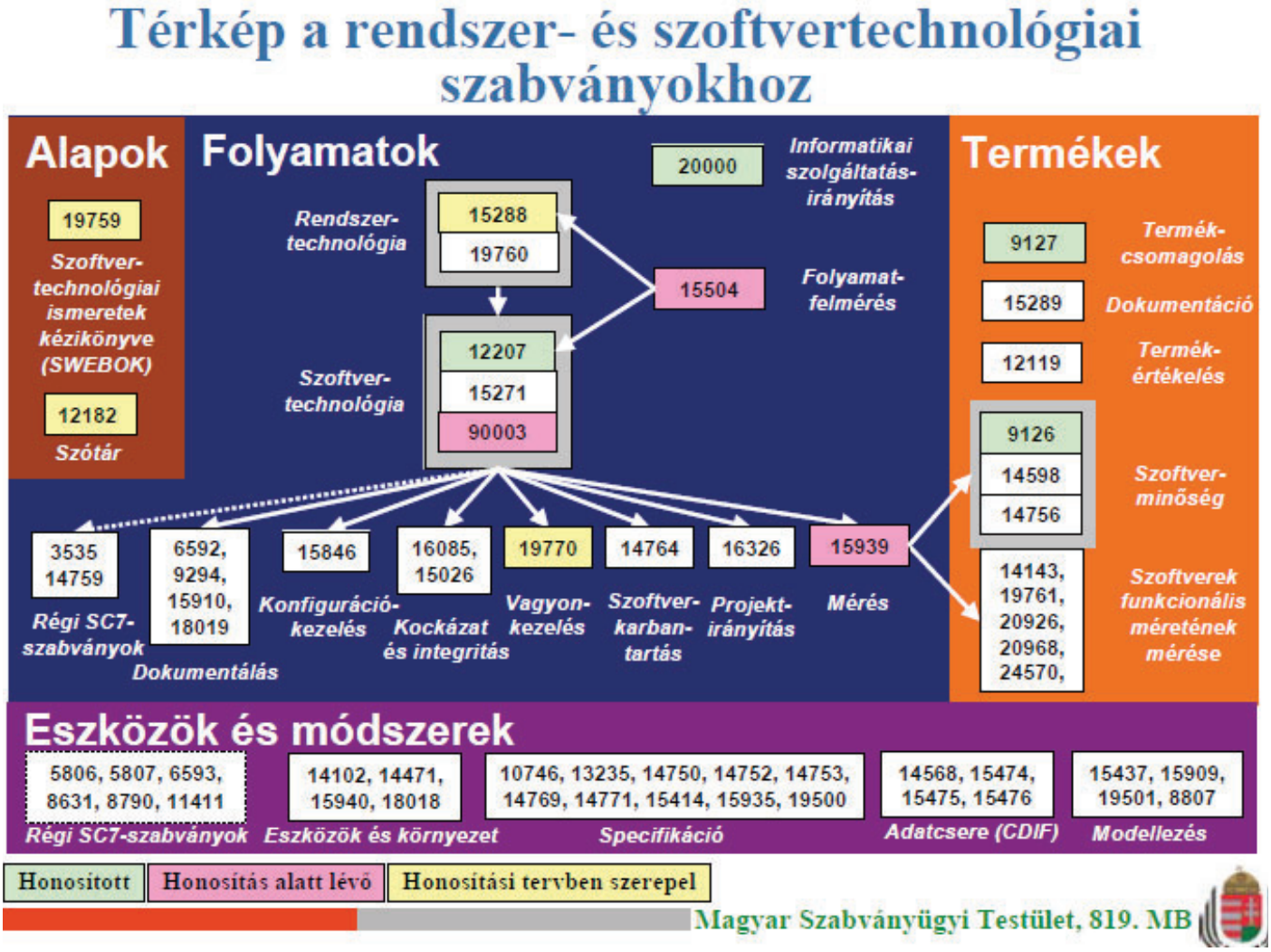
Azt a folyamatot, amely során egy adott probléma felmerülésétől, annak feltárásán, elemzésén, modelljének kialakításán keresztül, egy - a felhasználói igényeket kielégítő, adott feladat végrehajtást támogató - számítógéppel működtetett (szoftver)terméket hozunk létre, szoftverfejlesztésnek nevezzük.

Természetesen a fejlesztés sajátosságai alapján külön definiálható az információrendszer-fejlesztés, a webfejlesztés, a multimédia-fejlesztés.

A rendszerfejlesztők/szoftvermérnökök a rendszerszerű, fegyelmezett és számszerűsíthető módszereket helyezik előtérbe a szoftver fejlesztése, működtetése és karbantartása során egyaránt. E célból különböző modelleket és szabványokat fejlesztettek ki.

A Nemzetközi Szabványügyi Szervezet (ISO - International Organization for Standardization) által elfogadott szoftver életciklus-folyamatok szabványa az ISO/IEC 12207 szabványa.

Az ISO 12207 fő célja annak biztosítása, hogy a szoftverfejlesztésben részt vevő vásárlók, beszállítók, fejlesztők, karbantartók, üzemeltetők, vezetők és szakemberek közös nyelvet használjanak.



1.26. ábra Rendszer- és szoftvertechnológiai térkép

A szabvány eljárások egy halmazát tartalmazza, amelyek tevékenységek szerint vannak leírva. A tevékenységek feladatokra és eredményekre vannak felosztva. A szabvány Jelenleg 23 eljárást, 325 tevékenységet és 224 eredményt ír le.

Hasonló fejlesztési szabványok:

- Szoftvertermékek csomagolása: ISO/IEC 9127
- Szoftverminőség: ISO/IEC 9126
- **2015-ben megújult a folyamatszemlélet ISO 9001:2015 szabvány**
- **szintén 2015-ben megújul az ISO 50001 energiagazdálkodási irányítási rendszer szabványa**
- Mérési folyamat: ISO/IEC 15939
- Folyamatfelmérés: ISO/IEC 15504 (SPICE szabvány)
- CMMI alapja

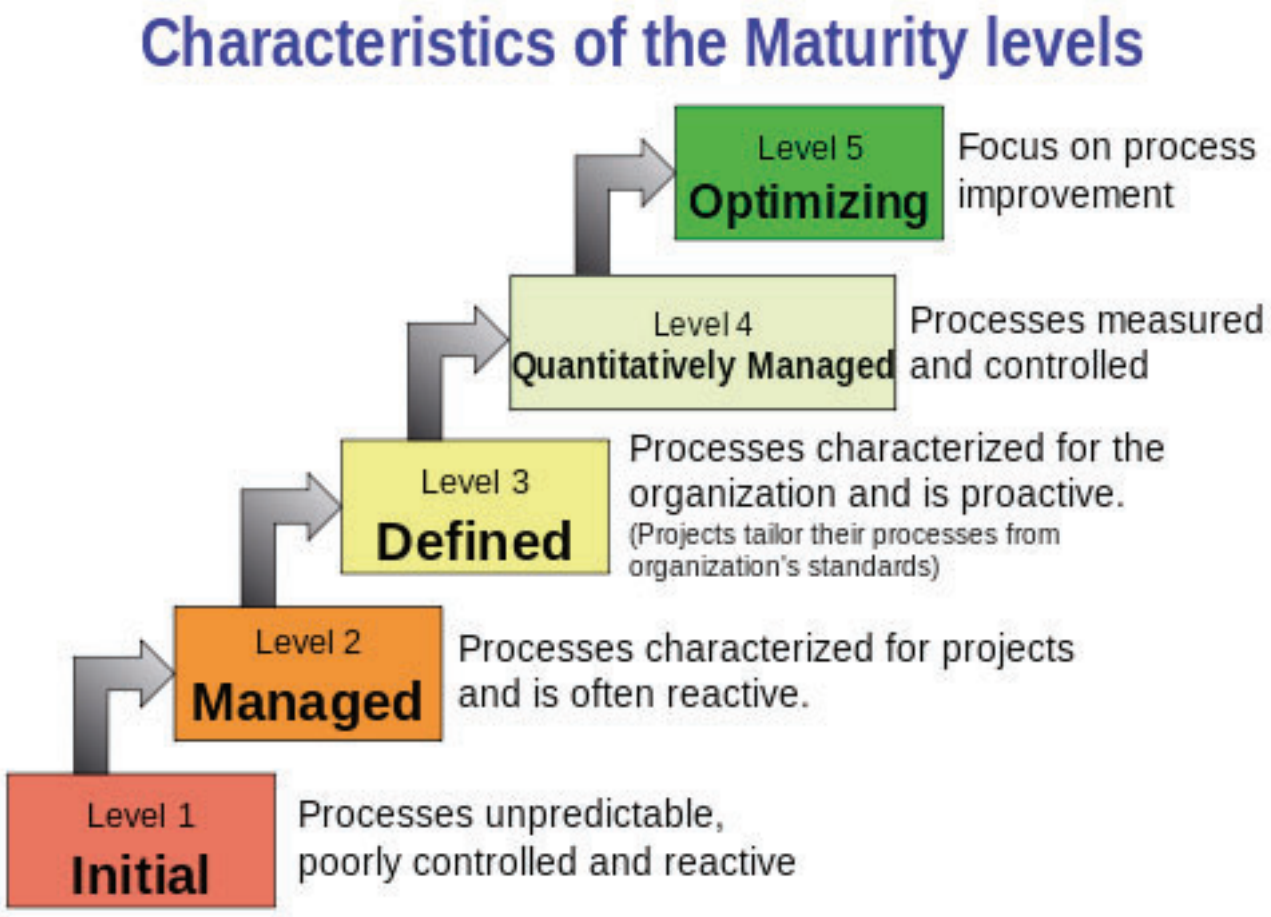
1.27. ábra Szabványok

Integrált Képesség Érettség Modell (CMMI - Capability Maturity Model Integration)

Ezt a modellt az amerikai Carnegie Mellon Egyetem Szoftverfejlesztési Intézete fejlesztette ki Pittsburgh-ben. Az érettségi modellt úgy lehet legjobban meghatározni, mint azon strukturált elemek gyűjteményét, amelyek leírják a szervezetben az érettség bizonyos szempontjait, és amely segít meghatározni és megérteni a szervezet folyamatait.

A CMMI egy módszer a szervezet szoftverfejlesztési folyamatainak teljes futamidejű értékelésére és felmérésére. A CMMI szoftverfolyamat-fejlesztési modell, mely két megközelítésben, lépcsősen és állandóan mutatja meg az IT folyamatokat.

Lépcsőzetes megközelítés:
1. szint: kezdeti (Initial)
2. szint: menedzselt/irányított (Managed)
3. szint: meghatározott (Defined)
4. szint: mennyiségileg menedzselt/irányított (Quantitatively Managed)
5. szint: optimalizáló (Optimizing)



1.28. ábra CMMI lépcsős megközelítése - LEFORDITANI

Folytonos megközelítés:
0. szint: nem teljes, befejezetlen (Incomplete)
1. szint: végrehajtott (Perfomed)
2. szint: menedzselt/irányított (Managed)
3. szint: meghatározott (Defined)
4. szint: mennyiségileg menedzselt/irányított (Quantitatively Managed)
5. szint: optimalizáló (Optimizing)

A CMMI (a) felépíti az integrált modelleknek az első csomagját, (b) növeli a legjobb gyakorlatot, a forrásmodellekből levont tanulságok alapján és (c) létrehoz egy keretet a jövőbeli integrációs modellekhez.

A módszer befolyást gyakorolt a magyar szabványként is megjelent ISO/IEC 12207:2008 szoftver életciklus-folyamatok szabványára, valamint az ISO/IEC 15504 szoftverfolyamat értékelési szabványtervezetre.

A CMMI 22 folyamatot azonosít, ezek a következők:
- projektmenedzsment (project management),
- mérnöki, fejlesztési(Engineering),
- folyamat menedzsment (Process management),
- a fejlesztést támogató folyamatok (Support).

Agilis módszertanok

Az agilis módszertanok egy szoftvercsomag kidolgozása során használt különböző módszereket, folyamatokat írják le.

Ez magában foglalja:
- projektmenedzseri folyamatot, amely a rendszeres ellenőrzést és átdolgozást ösztönzi,
- a vezetés filozófiáját, amely a csapatmunkát ösztönzi,
- az önszerveződést és elszámolhatóságot,
- a legjobb mérnöki gyakorlatok egy sorát, mely lehetővé teszi a kiváló minőségű szoftverek gyors átadását és
- egy üzleti szemléletet, amely összehangolja a fejlesztést a felhasználói igényekkel és a vállalat céljaival. Az agilis módszertan kiemelten foglalkozik a szoftverrel, mint a folyamat mértéke. Az agilis módszer kihangsúlyozza a munkaszoftvert, mint folyamatban lévő intézkedés.

1.29. ábra A 15 évvel ezelőtti kiáltvány

1.6.2. A korszerű műszaki architektúra fejlesztések hatása

A két-, illetve három rétegű kliens-szerver megoldásban az alkalmazás a szerverek egy halmaza által nyújtott szolgáltatás, amit a kliensek egy halmaza vesz igénybe. A kliensek ismerik a szervert, de a szervernek nem kell tudnia, hogy kik az ügyfelei. A kliensek és a szerverek valójában logikai folyamatok.

A szintek vagy rétegek

Megjelenítési réteg feladata, a számítás eredményeit megjelenítse a felhasználóknak, és összegyűjtse a felhasználói inputokat. Az alkalmazói réteg az alkalmazás-specifikus funkciókkal foglalkozik, az adatkezelési réteg feladata pedig az adatbázis rendszer menedzselése.

A szoftverfejlesztésben a többrétegű (gyakran n-szintűnek nevezett) architektúra, egy kliens-szerver architektúra, amelyben a megjelenítés, az alkalmazás és az adatkezelés logikailag különálló folyamatok. Az n-szintű alkalmazások felosztják a rendszert átfogó funkciókat az egyes rétegek (szintek) között.

Egy átlagos implementáció legtöbb esetben az alábbi rétegekkel találkozunk: megjelenítési réteg, logikai réteg, adathozzáférési, és adatbázis réteg. Egyes esetekben előfordul, hogy a különböző rétegek további alrétegekre vannak felosztva. Az egyes rétegeket egymástól elkülönülve fejleszthetjük, ha az egyes rétegek képesek lesznek egymással kommunikálni, és betartják a specifikációkban meghatározott szabványokat.

Az n-szintű alkalmazás lehetővé teszi, hogy az egyes rétegek, „fekete doboz" módon kezeljenek más rétegeket. Ez azt jelenti, hogy a rétegeket nem érdekli, hogy a többi réteg hogyan dolgozza fel az információt, mindaddig, amíg az adatok továbbítása a rétegek között a megfelelő formátumban történik.

Szolgáltatásorientált architektúra

A szolgáltatásorientált architektúra manapság a leginkább elterjedt modell. A szolgáltatásorientált architektúrában vesszük a szoftver szabványos n-szintű modelljét, a *megjelenítési*, *a logikai* és *adatkezelési* rétegekkel, és egy újabb réteget, az úgynevezett *szolgáltatási* réteget illesztünk be a *megjelenítési*, *a logikai* rétegek közé.

Csakúgy, mint az n-szintű architektúrában, a logika réteg megakadályozza a közvetlen függőségét az adatbázisban, a szolgáltatási réteg pedig megakadályozza a megjelenítés réteg közvetlen függőségét a logikai rétegtől. A megjelenítést elválasztva az adatábrázolástól, megengedi, hogy változtatásokat végezzünk az adatbázisban a megjelenítés befolyásolása nélkül, és hasonlóképpen, elválasztva a megjelenítést az üzletszabályzatoktól, lehetővé válik a funkcionalitást biztosító alkalmazások módosítása vagy cserélése az architektúrában, a megjelenítési réteg befolyásolása nélkül.

Az örökölt nagyszámítógép rendszerfejlesztésen alapuló kiterjesztése és integrációja

Örökölt rendszerek beazonosítják a régi rendszert, hogy a helyébe lépjenek. Az örökölt rendszer üzemen kívül helyezése egy nehéz folyamat. A régi rendszer élettartama során sok komponens jött létre, és ezeket alapos körültekintéssel kell eltávolítani, majd néhány összetevőt átírni az új rendszerbe. Az alkalmazott bevezetési modelltől függően a régi rendszer leszerelése egybeeshet az új rendszer éles indításával, vagy kezdődhet az indítás egy későbbi időpontban, illetve párhuzamosan futhat mindkettő, miközben fokozatos csere, vagy kísérleti bevezetés mellett. A forgatókönyv szerint a régi rendszer a teljesítményét akkor is lehet csökkenteni, amikor az új rendszer teljesítménye már megfelelő.

A régi rendszer általában tartalmaz néhányat (vagy az összeset) az alábbi dokumentumok közül:

- szabványok és rendszerdokumentáció, az eredeti rendszertervvel és az ajánlatkérési dokumentációval
- beszállítói szerződések és verzió történet
- rendszer átvizsgálási dokumentumok és vizsgálati eredmények
- a jelenlegi rendszer adatbázisa és archivált adatbázisok forráskóddal, adatokkal és biztonsági másolatokkal együtt
- A felhasználói hozzáférés, engedélyek és biztonsági adatok
- Rendszer jelentések
- Képzési anyagok és felhasználói kézikönyvek
- A hardver és szoftver elemek teljes jegyzéke, minden olyan kisegítő funkcióval együtt, hogy ha a régi rendszert kikapcsolják, a szervezetnek egészen biztos, hogy többé ne legyen szüksége rá.

1.6.3. Összetett rendszerek komplexitása, a komplexitás menedzselése

A „rendszerek rendszere" egymástól függő rendszerek halmaza vagy elrendezése, amelyek egy adott képesség előállítása érdekében kapcsolódnak egymáshoz. Bármely részrendszer kiesése jelentősen rontja az egész rendszer teljesítményét, képességeit. Az összetett rendszerek esetében néha úgy előfordul, hogy az egyes komponenseinek viselkedése nehezen kiszámítható. Ezt a jelenséget nevezik *emergens* viselkedésnek.

A *rendszerek rendszere* fejlesztés tervezéssel, elemzéssel, rendszerezéssel foglalkozik, és integrálja a már meglévő és új rendszerek képességeinek keverékét úgy, hogy az összetett rendszerének képessége nagyobb legyen, mint az egyes összetevők képességének összege.

A rendszerek rendszerének egy példája egy gyártási szervezet, amely termelésre, élelmiszer- és üdítőital termékekre specializálódott. Ez a szervezet a gyártóüzemében egy tételenként előíró rendszer segítségével állítja elő a termékeket, része a raktározási rendszer, amely fogadja a beszállított összetevőket, és a szállítási rendszer, ami elküldi a kész termékeket az ügyfeleknek világszerte, a szállításokat az ügyfél szolgáltatási rendszer nyomon követni, és végül a pénzügyi rendszer számláz az ügyfeleknek. Egy összefüggő rendszerben minden ilyen integráció nagyon bonyolult és költséges, nehéz és ijesztő feladat, amely sok szakmai tapasztalatot követel.

Az ilyen rendszerek összetettsége és együttműködtetése olyan szakképzett IT dolgozókat igényel, akik képesek átlátni és menedzselni a teljes rendszert. A növekvő összetettség és a szakképzett IT szakemberek hiánya folytán elkerülhetetlenné válik a számítástechnikához kapcsolódó funkciók jelentős részének automatizálása. Ez a probléma egyre inkább előtérbe kerül, és a technológiától való függésünk exponenciális növekedését eredményezi. Az egyik megoldási javaslat, hogy olyan számítógépes rendszereket hozzanak létre, amelyek önmagukat szabályozzák ugyanúgy, mint ahogy a vegetatív idegrendszer szabályozza és védi a testünket. Mindez radikális változást hoz az üzleti világban, az egyetemek életében, és a világ kormányzatainak számítógépes rendszereinek tervezésében, fejlesztésében, kezelésében és karbantartásában.

Egy aktuális javaslat a szervezetek számítástechnikai komponenseinek olyan hálózata, amely kielégíti a szükségleteinket, amikor azok felmerülnek anélkül, hogy lelki vagy akár fizikai erőfeszítést kellene tennünk. A hangsúly áttevődik a „számítógépről" az „adatokra". Autonóm rendszer magas szintű belső szabályai alapján önálló döntésekre képes; folyamatosan ellenőrzi és optimalizálja saját állapotát, és automatikusan alkalmazkodik a változó körülményekhez.

Az autonóm számítástechnika egy teljesen új kutatási területet és újszerű üzleti tevékenységet követel. Mivel az információhoz való hozzáférés egyre inkább PC-n, kézi- és vezeték nélküli eszközökön keresztül történik, a jelenlegi infrastruktúra, a rendszerek és adatok stabilitása egyre inkább ki van téve az üzemszünetek és általános elhanyagoltság veszélyének. Sokan feltételezik, hogy általában a számítástechnika, a hozzá kapcsolódó infrastruktúra, a köztes rétegek, és az azokat fenntartó szolgáltatások fejlesztésében hamarosan el fogunk érni egy kritikus küszöböt. A rendszerek növekvő komplexitásának szintje meghaladja azt, amit az ember képes biztonságosan kezelni.

A több adathoz való hozzáférés, az elosztott források, a hagyományos központosított tárolóeszközök biztosítják a felhasználóknak az információk átláthatóságát és elérését, ott és akkor, amikor szükségük van rá. Ugyanakkor az új számítógépes szemlélet szükségessé teszi, hogy a fejlődő elosztott hálózatok egyre inkább önmenedzselő, öndiagnosztizáló, és ugyanakkor a felhasználók számára átlátható ipari megoldásait keressük.

Az új számítógép paradigma azt jelenti, hogy a számítógépes rendszerek, szoftverek, tároló eszközök tervezésében és megvalósításában a felhasználói igények maximális figyelembe vétele mellett a fenti alapelveket kell követni.

Az öngyógyítás, önkonfiguráció, önoptimalizálás és önvédelem azok a tulajdonságok, amik egy rendszert autonómmá tesznek.

2. Adatmenedzsment és adatbázisok

2.1. Adatok és tranzakciók

Tanulási célok

- Vázolja, hogy miért fontos az adatok tartós tárolása a tranzakció-feldolgozás és a jelentéskészítő rendszerek részére.
- Bemutatja, hogy az atomiság, a konzisztencia, az izoláció és a tartósság hogyan működik közre a biztonságos adatbázis-tranzakciók szavatolásában.
- Vázolja egy többfelhasználós rendszer tervezési és fenntartási feladatait, mint a redundancia és inkonzisztencia, az integritási problémák, az adatok rugalmassága, a konkurens hozzáférés és biztonság.

2.1.1. Tranzakció-feldolgozó rendszerek

A *tranzakció-feldolgozó rendszer* olyan típusú információs rendszer, amely a szervezet ügyleteinek (tranzakcióinak) adatait gyűjti, tárolja, módosítja és teszi elérhetővé. A szervezet kereskedelmi sikere a tranzakciók megbízható feldolgozásától függ: ez biztosítja a vevői megrendelések időben való teljesülését és a partnerek (például beszállítók, vevők) kifizetését, illetve hogy fizethessenek. A tranzakció-feldolgozó rendszerek az ügyek gyors feldolgozására kínálnak eszközöket, biztosítva az adatok zavartalan áramlását és a folyamatok előrehaladását a teljes szervezetben. Ezeket a rendszereket arra tervezik, hogy gyakorlatilag azonnal feldolgozzák a tranzakciókat, és ezzel biztosítják, hogy a belőlük származó adatok elérhetőek legyenek az azt igénylő eljárások számára.

Nyilvánvalóan nem várható el az ügyfelektől, hogy a hibákat elnézzék, tehát a tranzakció-feldolgozó rendszereket úgy kell kialakítani, hogy az ügyletek ne legyenek hozzáférhetőek illetéktelenek számára, emellett a rendszer maga folyamatosan működőképes maradjon. Az ügyeket minden alkalommal ugyanúgy, a hatékonyság maximalizálásával kell feldolgozni. Ennek biztosítása érdekében a tranzakció-feldolgozó rendszerek kezelőfelületeit úgy alakítják ki, hogy mindegyik tranzakcióhoz azonos adatokat kérjenek be az ügyfél személyétől függetlenül.

A tranzakció-feldolgozó rendszerekkel szemben felmerülő igények:

- egyidejűleg több száz vagy akár több ezer felhasználót kezeljenek;
- lehetővé tegyék, hogy egyszerre több felhasználó ugyanazokkal az adatokkal dolgozzon azonnali adatfrissítéssel;
- a hibákat biztonságos és következetes módon kezeljék.

2.1.2. ACID követelmény

A tranzakció-feldolgozó rendszerek nagy mennyiségű adattal dolgoznak, melyek tárolására *adatbázis*t használnak. Az adatbázis olyan módon tárolt adatok együttese, amely lehetővé teszi nagy mennyiségű adat gyors kezelését (visszakeresés, bővítés, módosítás, kigyűjtések, adatvédelem) az adatok közötti összefüggéseket is figyelembe véve. Az adatbázisban a szervezet egyes ügyleteit követő adatmozgások, adatváltozások legtöbbször több egymást követő művelet eredményeképpen jönnek létre. Az ilyen összetartozó műveletsorozatokat *adatbázis-tranzakció*knak nevezzük, melyeknek meg kell felelni az úgynevezett *ACID* követelménynek. Az ACID mozaikszó az alábbi négy elvárásra utal.

Atomiság (Atomicity)

Az atomiság azt jelenti, hogy a tranzakció vagy teljes egészében végrehajtódik vagy egyáltalán nem. Ha valamilyen hiba miatt a tranzakció teljes végrehajtása meghiúsulna, akkor vissza kell állítani a kiindulási állapotot. **Példa**: egy összetett, több szabályt alkalmazó árleszállítási akció nem végződhet úgy, hogy nem kapja meg minden áru a megadott szabályrendszernek megfelelő új árat, mert nehéz (vagy nem is lehet) kibogozni, hogy melyiknél van már az új ár, és melyiknél még a régi.

Konzisztencia (Consistency)

Az adatbázis adathalmazának ellentmondásmentesnek kell lenni, és az adatbázis által leírt valóság szabályainak meg kell felelnie. (Például a termékek ára csak pozitív szám lehet.) Az utóbbit az úgynevezett *integritási szabályok* biztosítják. A tranzakciókra vonatkozó konzisztencia követelménye azt írja elő, hogy a tranzakció végrehajtása ezt az állapotot nem ronthatja el. **Példa**: egy árleszállítás alkalmával az új árakat beállító műveletsor nem eredményezhet egyetlen árunál sem 0 vagy negatív értéket.

Izoláció (Isolation)

Az adatbázist egyszerre több felhasználó is használja, tehát egyszerre többen is indíthatnak valamilyen tranzakciót. Az izoláció követelménye azt írja elő, hogy az egyes tranzakciók végrehajtása nem befolyásolhatja egy másik, folyamatban lévő tranzakció kimenetelét. Tehát az elindított tranzakciónak olyan eredménnyel kell végződnie, mintha az csak egyedül futott volna. **Példa**: egy árleszállítási művelet nem érintheti a már megkezdett eladási folyamat eredményét.

Tartósság (Durability)

A tartósság azt jelenti, hogy az egyszer már végrehajtott tranzakciók eredménye nem veszhet el. Ezt egyrészt biztonságos tárolási technikákkal, másrészt a *naplózás*sal – mely minden elvégzett műveletet dokumentál – biztosítják. **Példa**: áramszünet miatt egy lezajlott vásárlás adatai nem veszhetnek el.

Ez a négy feltétel biztosítja azt, hogy a tranzakció-feldolgozó rendszerek tervszerű, szabványosított és megbízható módon teljesítsék tranzakcióikat.

2.1.3. Tervezési és fenntartási szempontok

Az **adatredundancia** vagy egyszerűen *redundancia* jelentése: adatok többszörös tárolása egy rendszeren belül. A redundancia egyrészt pazarló mind a tárolás és mind az adatkezeléshez szükséges idő szempontjából, másrészt egymásnak ellentmondó adatokhoz vezethet. Az utóbbi bekövetkezésekor mondjuk, hogy az *adathalmaz inkonzisztens*. Ez az állapot bizonytalanságot szül: melyik a legutolsó vagy a leghitelesebb változat?

Példa redundanciára, ha az eladások adatai között tároljuk az azt intéző munkatárs nevét is (g több eladás ugyanattól a személytől, többször tárolt név). Ha valamelyik kollégának megváltozik a neve (mondjuk házasság kapcsán) minden eddigi eladásnál módosítani kell a nevét (g időpazarlás), vagy ha ezt nem tesszük meg, akkor adataink azt az információt fogják hordozni, hogy két különböző személy végezte az ő eladásait g ellentmondó (inkonzisztens) adatok.

A fentiekből következik, hogy a redundanciát minimalizálni kell, az inkonzisztenciát pedig kiküszöbölni.

Az **adatintegritás** problémaköre az adatatok helyességével és ennek fenntartásával foglalkozik. Ennek érdekében szabályokat lehet megfogalmazni (integritási szabályok), melyek célja, hogy megakadályozza a nem odaillő adatok létrejöttét, akár adatrögzítés során, akár műveletvégzés eredményeként. Például a teljesítési határidőnek csak jövőbeli dátumot lehessen adni.

A **konkurens hozzáférés** azt a lehetőséget jelöli, hogy egyszerre több felhasználó tud ugyanazzal az adathalmazzal dolgozni. **Példa**: több egyidejű jegyfoglalás vagy eladás és áru-bevételezés egy időben.

A **biztonság** az az elvárás, hogy a rendszert felépítése, folyamatai és az eljárásai megvédjék a nem rendeltetésszerű tevékenységektől. A nem rendeltetésszerű tevékenységek a következő típusokba sorolhatóak: visszaélések, rosszindulatú támadások és nem szándékos hibák. Az utóbbiak engedélyezett egyénektől vagy eljárásoktól származnak. Az adatbázis-biztonság a számítógép-biztonság szélesebb témakörének egy részterülete.

A fent leírtak mindegyikét figyelembe kell venni a rendszer kialakításakor, azonban a fenntartási időszakban sem szabad megfeledkezni róluk, például a rendszer módosításai esetén. A módosításokat a rendszerben esetlegesen talált hibák vagy a rendszer teljesítményének javítása, illetve a megváltozott környezethez való átdolgozások teszik szükségessé.

2.2. Adatbázis-szerkezet

Tanulási célok

- Különbséget tesz a fájlkezelő rendszer és az adatbázis-kezelő rendszer (DBMS) között.
- Bemutatja az adatbázis-rendszerek komponenseit, mint például adatállományok, adatszótárak, indexek, statisztikai adatok.
- Vázolja a gazdasági élet azon területeit, ahol adatbázis-kezelő rendszert használnak, és felismeri az azok által kínált előnyöket.
- Vázolja az adatbázis-kezelő rendszerek komponenseit, mint a lekérdező nyelv, a jelentésgenerátor, az adminisztrációs eszközök, a konkurenciavezérlők, a tranzakció-kezelő és a biztonsági mentés/helyreállítás eszközei.
- Bemutatja az adatbázis-adminisztrátor, az adatbázis-tervező/programozó és az adatbázis-felhasználó szerepeket.

2.2.1. Adatbázis-kezelő rendszerek

A *fájlkezelő egy* olyan egyszerű rendszer, amely lehetővé teszi a fájlok tárolását és elérését. A fájlkezelő rendszerben létrehozhatunk, megváltoztathatunk állományokat, lekérdezhetjük a tulajdonságaikat, de egyszerre csak egy fájllal végezhetünk műveletet. A fájlkezelőkben jellemzően nincs lehetőségünk az adatkezeléshez programozásra.

Az *adatbázis-kezelő rendszer (DBMS, azaz Database Management System)* az adatbázisban tárolt információ tárolására, gyors kikeresésére, elérésére és módosítására szolgáló rendszer. Az adatbázis-kezelő nem maga az adatbázis, hanem a kialakítására tartalmához való hozzáférésre használt programrendszer. Az adatbázist fizikailag egy vagy több fájl alkotja, azonban hogy fizikailag hol helyezkedik el a háttértárolón, az számunkra lényegtelen, hiszen a fájlok tartalmához közvetlenül nem férhetünk hozzá: a benne tárolt adatok elérését az adatbázis-kezelő biztosítja.

Az adatbázis-kezelő határozza meg az adatok tárolásának és elérésének módját, foglalkozik a biztonság, az integritás, az adatok közötti konzisztencia, a válaszidő és a memóriaszükséglet problémáival. A beérkező kérések alapján utasítja az operációs rendszert a megfelelő adatok szolgáltatására. Az adatbázis-kezelők sokszor egy alkalmazásfejlesztésre szolgáló saját programnyelvet is szolgáltatnak, azonban az is előfordul, hogy erre a célra külső, hagyományos programnyelven írt kód integrálására adnak lehetőséget.

Az adatbázis-kezelő alkalmazása révén egy cég információs rendszere az éppen aktuális követelményeknek megfelelően rugalmasan módosítható, hiszen az adatbázis könnyedén bővíthető új adatelemekkel anélkül, hogy a már meglévő részeit meg kellene változtatnunk.

Ma legszélesebb körben a relációs adatbázis-kezelőket használják, melyek az adatokat táblák rendszerébe szervezik.

Az adatbázis-kezelők néhány előnye a fájlszemléletű adatkezeléshez képest:

- könnyebb az adatintegritás fenntartása;
- könnyebb különböző típusú adatokat érintő lekérdezések futtatása;
- könnyebb az adatbiztonság fenntartása.

2.2.2. Adatbázis-komponensek

Adatállományok

Az adatállományok tartalmazzák azokat az információkat, melyek tárolására létrehozták az adatbázist, míg a többi állomány, mint például az indexállományok és adatszótárak, adminisztratív információkat, úgynevezett metaadatokat tárolnak. Az adatállományok az információk elkülöníthető egységeit (az adatokat) valamilyen erre a célra kialakított struktúrában tárolják.

Adatszótár

Az adatszótár írja le az adatbázis szerkezetét: tartalmazza az adatbázisban lévő állományok listáját, az adatelemeiről szóló információkat (jelentésük, eredetük, használatuk, formátumuk, másik adattal való kapcsolatuk, kódolási módjuk…). Az adatszótár tartalmazza az adatok értelmezéséhez és kezeléséhez szükséges információkat.

A relációs adatbázisok esetén idetartozik például a táblák és mezők neve és leírása, a mezők adattípusa és hossza, a táblákban levő rekordok száma.

A jól kialakított adatszótár segíti az összetett adatbázis egészét átfogó konzisztencia kialakítását és fenntartását. Az adatszótár azonban a felhasználók számára nem elérhető, mivel nagyon fontos, hogy még véletlenül se okozhasson senki kárt a tartalmában.

Indexek

Az indexek a keresési és rendezési műveletek sebességének javítására szolgálnak.

Relációs adatbázisban az indexelt mezők értékeinek rendezett listája – az érték mellett az azt tartalmazó rekord pozíciójával – külön táblákban tárolódik. Ezek az indexek. A rendezett értékeken jóval gyorsabb keresési technikát lehet alkalmazni, majd a megtalált érték melletti rekordpozícióval pedig rögtön meghatározható a keresett adat helye az eredeti táblában. A rendezés pedig egyszerűen a rekordoknak az indextáblában egymás után következő pozíciók szerinti kiolvasásával adódik. Persze ez a rendezés csak látszólagos, hiszen a rekordok fizikai helye nem változik. Ezt nevezik logikai rendezésnek.

Statisztikai adatok

Az idők során felhalmozott, a vezetői döntéseket segítő, célszerűen kigyűjtött, rendszerezett, elemzett, értelmezett és kifejtett adatokat statisztikai adatoknak nevezzük.

2.2.3. Adatbázis-kezelő rendszerek alkalmazása

Az adatbázis-kezelő rendszereket, jellemzően a relációs adatbázis-kezelő rendszereket, többnyire vállalatirányítási rendszerekben alkalmazzák. Az ilyen rendszereket elsősorban vállalatoknál, intézményeknél illetve más olyan területeken alkalmazzák, ahol nagy mennyiségű egyszerre ki- és beáramló adat kezelésére van szükség. A *vállalatirányítási rendszer* olyan szoftveralkalmazás, amely automatizálja és integrálja egy cég üzleti folyamatait.

Minden nagy szervezetnek szüksége van adatbázisra épülő rendszerre, amely biztosítja több felhasználó egyidejű hozzáférését az információkhoz. Például egy banki adatbázis-alkalmazást folyamatosan nagyon sokan használnak különböző hozzáférési lehetőségekkel: az ügyfelek információt kérnek (egyenleg ellenőrzése egy ATM-nél stb.); a személyzetnek nyilvánvalóan más és mélyebb betekintése van a tárolt adatokba, de nekik sem egyformán; a programozók kezelőfelületeket alakítanak ki, amelyekkel lehetővé teszik és szabályozzák az adathozzáférést.

Az adatbázis-kezelő rendszereknek azonban nem csak a nagy szervezetek az alkalmazói, hiszen ahol nagy mennyiségű információ gyors elérésére és kezelésére van szükség, ott a legcélszerűbb megoldást az adatbázisok jelentik.

Az adatbázis-kezelők előnyei

Az adatok központosítása – Az adatbázis-kezelő lehetővé teszi az összes adat egy helyen való tárolását szemben az adatok különböző fájlokban, különböző számítógépeken, még esetleg különböző helyszíneken való tárolással. Ez csökkenti a szükséges adatbevitelek és frissítések számát, így a hibalehetőségeket is.

Az inkonzisztencia és a redundancia elkerülése – Ideális esetben minden adattételt elég egyszer tárolni, így kiküszöböljük a redundáns adattárolást, és ezzel az inkonzisztencia lehetőségét is.

A **többszörös hozzáférés** egyfelől azt jelenti, hogy az adatokhoz egyidejűleg több felhasználó férhet hozzá, másfelől pedig azt, hogy ez a hozzáférés különféle módokon történhet (akár még többféle programnyelv használatával is).

Fokozott teljesítmény – Az adatbázis-kezelő lehetővé teszi, hogy gyors választ kapjunk az információkérésekre. Mivel az adatok egyetlen adatbázisban tárolódnak, az összetett kéréseket a rendszer sokkal gyorsabban tudja kezelni, mintha az adatokat különböző helyekről kellene begyűjtenie. A gyorsabb válasz jobb ügyfélszolgálatot tesz lehetővé, és a kérések feldolgozásához szükséges időcsökkenése javítja a produktivitást.

Rugalmasság – Az adatok és az azokat kezelő programok egymástól függetlenek, ezért a programok már működő részeit nem kell módosítani, amikor új adattételekkel bővítik az adatbázist, vagy a fizikai tárolás változik.

2.2.4. Adatbázis-kezelő rendszerek komponensei

A **lekérdező nyelvek** olyan számítógépes nyelvek, amelyeken az adatbázisnak szóló kérdéseket és kéréseket lehet megadni. A relációs adatbázisokhoz széles körben használt lekérdező nyelv az SQL (Structured Query Language). Az adatbázis-kezelő SQL értelmezője ellenőrzi az megadott lekérdezés helyességét, ha hibásnak találja, akkor hibaüzenetet ad, különben pedig elvégzi az abban kért adatkezelést (például kigyűjtést).

A **jelentésgenerátor** az adatbázisból kigyűjtött adatokat jól értelmezhető formában jeleníti meg. Segítségével szabályozhatjuk jelentéseink szerkezetét és megjelenési formáját.

Adminisztrációs eszközök:
- biztonsági és felhasználó-kezelő eszközök,
- tároláskezelő eszközök,
- teljesítményfigyelés,
- kapacitáselemzés.

A **konkurenciavezérlés** feladata, hogy az adatbázis konzisztenciáját komolyan veszélyeztető konkurens (egyidejű) hozzáférések problémáját megoldja. Például azzal, hogy zárol bizonyos egységeket, mert a zárolt egységekhez (pl. adat, rekord, tábla) nem férhet hozzá más tranzakció a zár feloldásáig.

A **tranzakció-kezelő** a tranzakció-naplózást és a helyes tranzakció-állapotot megvalósító rendszer. Egy adatbázis-kezelő környezetben a tranzakciók végrehajtásának több lépésre lehet szüksége, ezért a rendszernek naplóznia kell a történteket, hogy lehetővé tegye a tranzakció véghezvitelét vagy – ha az nem sikerül – a visszavonását (roll back).

Emberi vagy hardverhibából eredő, az adatbázis bármelyik részén keletkező kár esetén alapvető fontosságú, hogy az érintett adatokat minimális fennakadással helyre tudjuk állítani. Lehetőleg a felhasználók ne is értesüljenek a hibáról, és a hiba ne legyen kihatással azokra az adatokra, amelyek nem sérültek, így ezek folyamatosan elérhetőek maradjanak. Ezért az adatbázis-adminisztrátornak meg kell határoznia és adott esetben végre is kell hajtania a megfelelő **helyreállítási stratégiát**, amely rendszerhiba esetén ép állapotba állítja vissza az adatbázist. A **biztonsági mentések** rendjének kialakítása fontos része ennek a munkának. Számos eszköz áll rendelkezésre egy működésben lévő adatbázis biztonsági mentésére, a jelenlegi trend az online biztonsági mentési megoldások irányába mutat.

2.2.5. Adatbázisokkal kapcsolatos feladatkörök

Adatbázis-felhasználó

Az adatbázis-felhasználók többsége az adatbázist egy speciálisan arra fejlesztett alkalmazáson (szoftveren) keresztül éri el. Az alkalmazás kezelőfelülete egyrészt előre kialakított adatbeviteli, adatkérési, tranzakció-indítási lehetőségeket, kigyűjtéseket, jelentéseket nyújt a felhasználónak, másrészt irányított hozzáféréssel szabályozza a felhasználó jogosultságait. Az adott felhasználó számára tárgytalan részek elfedése nem csak véd a jogosulatlan adatelérésektől, hanem egyszerűbbé is teszi a legtöbbször igen összetett adatbázis használatát.

Azoknál az adatbázisoknál, amelyek használatához nem készül külön alkalmazás, az egyes összetevőkre megadott jogosultságokkal szabályozzák a felhasználók hozzáférését az adatokhoz és a végrehajtható műveletekhez. Megint más esetekben az adatokhoz való hozzáférés alkalmazáson keresztül és a közvetlenül is lehetséges, természetesen a felhasználói jogosultságoknak megfelelően.

Példaként tekintsük egy bank adatbázisát, ahol a felhasználók egyrészt az ügyfelek, akik számlát vezetnek a banknál, másrészt a bank dolgozói. Nyilvánvaló a különbség közöttük az adatokhoz való jogosultság terén, ámde az egyes dolgozók esetén is hasonló a helyzet: ahogy a különböző pultoknál más és más üggyel foglalkozó tisztviselőt találunk, mindegyikük feladatához más és más adathalmaz elérése szükséges. A színfalak mögötti banki dolgozók közül kiemelve az adatelemzőket, róluk feltételezhetjük, hogy saját készítésű lekérdezésekkel faggatják az adatbázist, míg az előzőleg említett felhasználók biztosan nem az adatbázis szerkezetét böngészve, hanem célirányos kezelőfelülettel érik el a nekik szükséges adatokat.

Adatbázis-tervező/programozó

Az adatbázis-tervező az a szakember, aki az adatbázis logikai szerkezetét kialakítja (megalkotja az adatbázis részletes adatmodelljét) és meghatározza a kivitelezéshez szükséges fizikai tárolási paramétereket. Az adatbázis-felhasználóknál említett alkalmazás elkészítése a programozók feladata természetesen az adatbázis-tervezővel szoros együttműködésben. Ezeket a feladatokat a kialakítandó rendszer bonyolultságától függően elláthatja akár egy személy is, de többnyire jól strukturált munkamegosztásban dolgozó csapat végzi.

Optimális esetben az adatbázis-tervező/programozó (csapat) csak a kialakításkor és az esetleges javítások, továbbfejlesztések esetén dolgozik az adatbázissal. Adatvédelmi okok azt is indokolttá tehetik, hogy ők ne is találkozzanak az adatbázis éles (valódi) adataival.

Adatbázis-adminisztrátor (DBA: Database Administrator)

Az adatbázis-adminisztrátor felelős az adatbázis fizikai kialakításáért, elérhetőségének, üzemidejének és – a megadott költségvetési megszorítások mellett –maximális teljesítményének biztosításáért. Ő felel az adatbázis biztonságáért az adatokhoz való hozzáférési szabályozások meghatározásával és a biztonsági mentési és helyreállítási eljárások kialakításával.

Az adatbázis-adminisztrátor nem használja az adatbázist (nem visz be és kérdez le adatokat, nem indít tranzakciókat), hanem a biztonságos működését tartja fenn.

Mint láttuk a felhasználók különböző csoportjai és a többi feladatkör szereplői is egészen eltérő hozzáféréssel dolgoznak az adatbázison, ehhez pedig elengedhetetlen, hogy a tevékenység a szerepkör vagy felhasználói csoport azonosításával kezdődjön (név, jelszó). Azonban gyakran biztonsági, elszámolási vagy erőforrás-gazdálkodási okokból a rendszer használatát a felhasználók tevékenységére bontott napló rögzíti – ilyen esetben nem elegendő a felhasználói csoport, hanem a konkrét személy azonosítása szükséges bejelentkezéskor.

2.3. Adatmodellezés

Tanulási célok

- Meghatározza az „adatabsztrakció" fogalmát, és bemutatja a különbséget a fizikai, a fogalmi (logikai) és a nézet (felhasználói) szint között.
- Megkülönbözteti az adatmodellek különböző csoportjait, mint az objektumalapú logikai modell, a rekordalapú logikai modell és a fizikai adatmodell.
- Bemutatja a rekordalapú logikai modellek, mint a hierarchikus és a hálós modell, alapelveit.
- Bemutatja az objektumalapú logikai modellek, mint az egyed-kapcsolat és az objektumorientált modell, alapelveit.

2.3.1. Az adatabsztrakció szintjei

Az adatbázis a felhasználók számára absztrakt képet nyújt az adatokat tároló rendszerről, ily módon elrejti az adattárolás, -létrehozás és -fenntartás módjának részleteit. A jól kialakított adatbázis elfedi a felhasználók számára fölösleges, bonyolult részleteket.

Az absztrakció legalacsonyabb szintje, a **fizikai szint** foglalkozik a fájlokkal, a mappákkal és a fizikai eszközökkel, amelyek a rendszert tárolják. Tehát az adatok tárolásának összetett struktúráit tartalmazza részletesen leírva.

A fizikai szint fölött következő absztrakciós szint a **fogalmi (koncepcionális)**, más néven **logikai szint**. Itt lehet az adatok logikai struktúráját leírni: milyen adatokat tárolunk, ezek tulajdonságait és a közöttük lévő kapcsolatokat. Ez a szint, amit az adatbázis-kezelő nyújt az adatszerkezet kialakítására, azoknak az objektumoknak a létrehozására, amelyekhez műveleteket biztosít.

Az adatbázis-kezelők lehetőséget biztosítanak akár nagyszámú felhasználó számára is az erőforrások (adatok és eljárások) megosztására, ami a sok különböző felhasználói igény összetettségéhez vezet. Például a vállalati adatbázis néhány HR felhasználójának a fizetési adatokat kell látnia, míg ugyanabból a munkakörből más felhasználóknak csak a munkateljesítmény vagy jelenlét bejegyzéseire van szüksége, és akkor még nem is beszéltünk a más munkakörökben dolgozó munkatársakról, akik ugyanazon adatbázisnak más-más részeit használják.

A fentiek alapján célszerű a különböző felhasználók számára az adatbázisról különböző nézeteket nyújtani. A **felhasználói szint** az absztrakció legmagasabb szintje, más néven a **nézetek szintje**, amely a különböző felhasználói csoportokkal az adatbázis számukra szükséges részeit láttatja. Ebből következik, hogy ugyanannak az adatbázisnak a felhasználói igényeknek megfelelően sok nézete létezhet.

2.3.2. Az adatmodellek típusai

Az adatmodell az adatok, a köztük lévő kapcsolatok, az adatszemantika és az adatmegszorítások leírására szolgáló fogalmi eszközök gyűjteménye. Az adatszemantika tulajdonképpen az adatok jelentése és használata. Egy adott adatbázis objektumainak jelentésén érthetjük azokat a valós világbeli objektumokat, amiket ezek reprezentálnak. Az adatmegszorítások olyan szabályok, amelyeket be kell tartani az adatbázisba való adatbevitelkor. Az adatok integritásának fenntartására szolgálnak.

Az adatmodelleket három különböző csoportba soroljuk: objektumalapú, rekordalapú és fizikai adatmodellek.

Az **objektumalapú logikai modellek** az adatbázist fogalmi szinten írják le és ésszerűen rugalmas strukturálási lehetőségeket nyújtanak, melyek révén a tervező az adatmegszorításokat explicit (közvetlenül kifejezett) módon határozhatja meg.

Néhány Példa:
- egyed-kapcsolat modell,
- objektumorientált modell,
- bináris modell,
- szemantikai adatmodell,
- funkcionális adatmodell.

A **rekordalapú logikai modellek** az adatbázist szintén fogalmi szinten írják le. Elnevezése a felépítés alapstruktúrájából ered. A *rekord* egy összetett adattípus: különböző típusú adatok (ezek a rekord mezői) rögzített számú és sorrendű együttese. A rekordot alkotó mezőknek nem csak az adattípusa, hanem általában a hossza (bájtméret vagy karakterek száma) is rögzített.

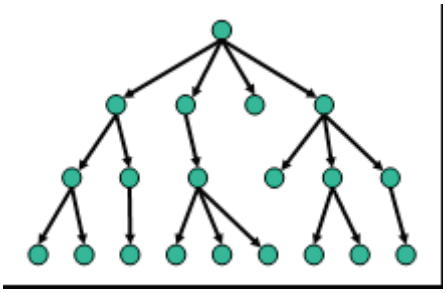
A leginkább elterjedt rekordalapú adatmodellek:
- relációs,
- hálós,
- hierarchikus.

A **fizikai adatmodellek**et az adatok tárolási szinten való leírására használják. Csak nagyon kevés ilyen modellt használnak.

2.3.3. Rekordalapú adatmodellek

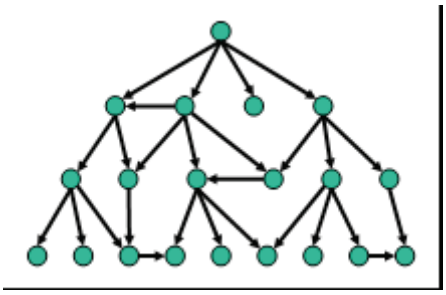
A rekordalapú logikai modellek a valós világ objektumait (például termékeket, számlákat, partnereket) ezek adatait magukba foglaló rekordokkal reprezentálják. Ennek megfelelően az adatbázisban tárolt adatok rekordok halmazába szerveződnek.

A **hierarchikus modell** volt az első logikai adatmodell. A rekordok közötti kapcsolatokat fagráfok írják le. Ez az erősen kötött kapcsolati szerkezet csak nagyon korlátozottan tudja követni a valóság objektumainak összetett viszonyrendszerét.



2.1. ábra A fagráf szigorú hierarchiája

A **hálós modell** feloldja a hierarchikus modell kötöttségét, mert itt a kapcsolatokat tetszőleges gráfok írják le, vagyis bármilyen kapcsolódás megengedett. Azonban ennél a modellnél a kapcsolatrendszer hamar elburjánzik és bonyolulttá válik, ami nehézkessé teszi az adatbázis használatát.



2.2. ábra További kapcsolatok engedélyezése felbontja a fastruktúrát

A **relációs modell** gondolata a hierarchikus és a hálós modell alkalmazásainak nehézkes működése miatt született meg. A legkülönbözőbb helyzetekre való rugalmas alkalmazhatósága és a benne rejlő gyors működés lehetősége miatt nagyjából teljesen kiszorította két elődje alkalmazását. A mai napig ennek az adatmodellnek az alkalmazása a legelterjedtebb, ezért a következő teljes fejezet tárgyalja.

2.3.4. Objektumalapú adatmodellek

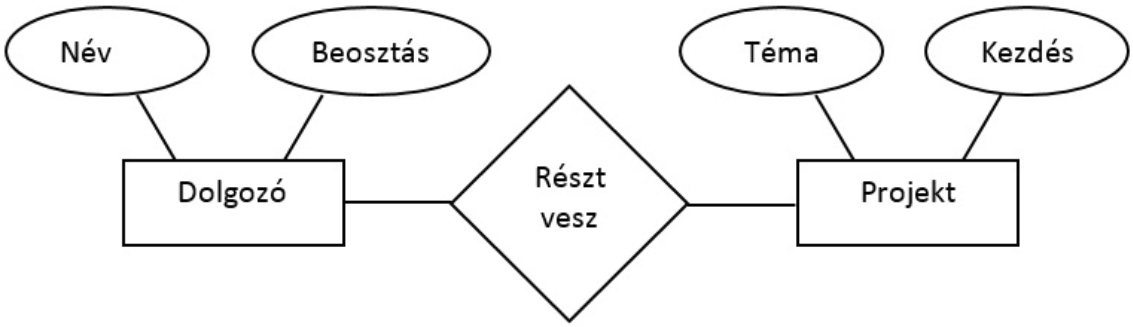
Az egyed-kapcsolat (EK vagy ER, azaz Entity-Relationship) modell

Az egyed-kapcsolat modell objektumokból és a köztük lévő kapcsolatokból felépülő világ észlelésén alapszik. Az egyed egy olyan létező objektum (termék, dolgozó, számla, projekt…), ami a többi objektumtól megkülönböztethető. Az egyedeket a tulajdonságaik (attribútumaik) írják le (például a termék neve, ára…; számla száma, kelte…; projekt témája, kezdete…). Az egyedtípus az ugyanolyan tulajdonságokkal jellemezhető egyedek halmaza (például egy vállalat alkalmazottai). Az egyedtípus minden egyede ugyanolyan tulajdonságokkal, de ezekhez saját értékekkel rendelkezik. Az egyedtípusokat névvel és tulajdonságlistával lehet megadni.

Az egyedek közötti kapcsolatok is általánosíthatóak egyedtípusok közötti kapcsolattípusokra, ám gyakrabban ezt is egyszerűen kapcsolatnak nevezik. Például, ha egy dolgozó részt vesz egy projektben, az egy kapcsolat, de általában a dolgozók részvétele a projektekben az már kapcsolattípus, de általában csak kapcsolatnak mondjuk.

Az EK-modell leírására az egyed-kapcsolat diagramot használják, mely az alábbi jelöléseket alkalmazza:

- egyedtípus – téglalap,
- tulajdonság (attribútum) – ellipszis,
- kapcsolat – rombusz.



2.3. ábra Példa *EK-diagramra*

Objektumorientált modell

Az objektumorientált programozás a valóságos világ objektumainak leképezésére törekszik.

Az objektumok értékeket tartalmaznak, amik változókban tárolódnak az objektumon belül. A rekordalapú modellektől eltérően ezek az értékek maguk is lehetnek objektumok. Az objektumok olyan eljárások kódjait is tartalmazzák, amelyek az adott objektumokkal/objektumokon végeznek műveleteket. Ezeket az eljárásokat metódusoknak nevezik. Az objektum belseje, a példányváltozók és metóduskód, nem láthatók külsőleg. Egy objektum csak úgy férhet hozzá egy másik objektum adataihoz, ha annak megfelelő metódusát meghívja, „üzenetet küld" az objektumnak.

Az azonos típusú értéket és azonos metódusokat tartalmazó objektumok csoportjai az osztályok, ezért nevezzük az objektumokat az adott osztály példányainak. Emellett azonban egy új objektum létrehozásánál (példányosítás) az osztály az objektumnak mintegy típusdefiníciójaként funkcionál.

Az objektumorientált adatbázis az objektumorientált programozási elveket és az adatok hosszú idejű tárolását kombinálja.

2.4. A relációs adatmodell

Tanulási célok

- Vázolja a relációs modell előnyeit, mint a redundancia mentesség, a rugalmasság és a skálázhatóság.
- Bemutatja relációs modell főbb alapfogalmait, mint a reláció, a kulcs, az elsődleges kulcs, az alternatív kulcs, az idegenkulcs és a hivatkozási integritás.
- Kifejti egyszerű példákon keresztül a normalizálás folyamatát az első, második és harmadik normál formára.

2.4.1. A relációs modell alapfogalmai

Reláció

A relációs modell tárgyalásakor az egyed-kapcsolat modellnél bevezetett fogalmakat (egyed, egyedtípus, tulajdonság, kapcsolat) itt is azonos értelemben használjuk. A relációs modellben az adatokat és a kapcsolatokat *táblák* hordozzák, egyedtípusonként külön-külön tábla és bizonyos kapcsolatokat is külön tábla.

Táblán egy olyan táblázatot értünk, mely szigorúan sorokra és oszlopokra tagolt, minden sor azonos felépítésű, és minden oszlop egyedi névvel rendelkezik a táblán belül. Mivel a sorokat különböző típusú adatok rögzített sorrendű együttese alkotja, tehát a sorok rekord típusúak, ezért a táblák sorait gyakran a *tábla rekordjai*ként szokták említeni. A táblák oszlopait pedig, mivel az egyes rekordok azonos mezői alkotják, a *tábla mező*nek is nevezik.

Az imént meghatározott tábla fogalom megfelel a matematika reláció fogalmának, ami nem véletlen, hiszen E. F. Codd, a relációs modell megalkotója, a relációs algebrára építette elméletét. Innen származik a *relációs modell* és a *relációs adatbázis* kifejezés. Az adatbázis-kezeléssel foglalkozó írások sokszor a tábla szó helyett a *reláció* kifejezést használják.

A következő táblázat a modellezni kívánt valóságra utaló fogalmakat és a nekik megfelelő relációs adatbázis-struktúrákat párosítja.

Példák a valós életből	A példákat összefogó fogalom	Az adatbázis megfelelő fogalma
Termék, számla, dolgozó, projekt	Egyedhalmaz/egyedtípus (egyes irodalmakban egyed)	Reláció/tábla
Terméknév, számla kelte, név, a projekt vége	Tulajdonság/attribútum	A tábla oszlopa/mezője
Ez a könyv, egy kiállított számla, én, a EUCIP Core tananyag adaptációja	Egyed (egyes irodalmakban egyed-előfordulás)	A tábla sora/rekordja, mely az adott egyed adatait tartalmazza
EUCIP Core tankönyv, 2015.07.31., Rétsághy Zsolt, 2015.04.30.	Egy egyed valamelyik tulajdonsága	Egy rekord valamelyik adata/mezője

Kulcs (key)

A relációs adatbázis a táblák sorainak (rekordjainak) kezeléséhez kulcsokat használ. A táblának azokat a mezőit (oszlopait) nevezzük kulcsnak, amelyeket rendezésre vagy azonosításra használunk. Előfordul, hogy több mező együttesen szolgál egy ilyen célt, ilyenkor *összetett kulcs*ról beszélünk. Például a helységnevet és a helységen belüli címet később tárgyalt okból külön mezőben tároljuk, így a címszerinti rendezéshez összetett kulcsra lesz szükségünk.

Elsődleges kulcs (primary key)

A relációs adatbázisban minden rekordnak rendelkezni kell a táblán belüli egyedi azonosítással, ami vagy a természetes adatokon alapszik, vagy az erre a célra létrehozott azonosító. Például a dolgozók azonosítására használhatjuk a személyi igazolványuk számát (ez természetes adat, hiszen a mindennapi életben használt, létező dolog), de szolgálhat egy belső azonosításra kialakított törzsszám is. A táblán belüli azonosításra használt mező vagy mezőkombináció az elsődleges kulcs. Nyilvánvaló, hogy csak úgy töltheti be az azonosító szerepet, ha értéke mindegyik rekordban más és más. Miután kijelöltük egy tábla elsődleges kulcsát, az adatbázis-kezelő érvényre juttatja a kulcs egyediségét: nem enged már meglévő elsődleges kulcs értékkel új rekordot felvenni.

Alternatív kulcs (alternate key)

Az alternatív kulcs bármilyen elsődleges kulcsnak alkalmas, de nem arra kiválasztott kulcs. Például a már előbb említett törzsszám és személyiigazolvány-szám mellett még a név + születési dátum összetett kulcs is alkalmas lenne a dolgozók azonosítására, amit úgy mondunk, hogy mindhárom *kulcsjelölt*. Miután a törzsszámot választjuk ki elsődleges kulcsnak, a másik kettő válik alternatív kulccsá.

Idegenkulcs (foreign key)

Az elsődleges kulcs rendkívül fontos felhasználása, hogy segítségével az általa azonosított rekordra az adatbázis bármelyik rekordjából lehet hivatkozni. A relációs adatbázisban ezzel a módszerrel valósítják meg a rekordok, és ezzel az egyedek közötti kapcsolatokat. Az idegenkulcs az a mező vagy mezőkombináció, amely elsődleges kulcs értékeket tartalmaz, ám ezzel nem a saját rekordját, hanem egy kapcsolódó

rekordot azonosít. A kapcsolódó rekord általában egy másik táblában helyezkedik el, de az sem kizárt, hogy ugyanabban a táblában legyen. Például a „számla" táblában a „kiállító" mező dolgozók azonosítóit tartalmazza (minden számlánál azét, aki azt a számlát kiállította), azonban a „dolgozó" táblában a „főnöke" mező értékei ugyanebben a táblában található rekordokat azonosítanak (olyan dolgozókét, akik más dolgozók főnökei).

Az elsődleges kulcstól eltérően az idegenkulcs értékeinek nem kell egyedinek lenni, hiszen pl. több számlánál is lehet ugyanaz a kiállító, és több dolgozónál is ugyanaz a főnök.

Hivatkozási integritás

A hivatkozási integritás az a természetes követelmény, hogy az idegenkulcsban csak olyan értékek fordulhatnak elő, amelyek szerepelnek a hivatkozott tábla elsődleges kulcsában. Például nem szerepelhet a számla kitöltőjénél olyan törzsszám, amelyez nincs dolgozó. Az integritást megsértő *hivatkozási anomália* leginkább akkor következhet be a példánkban, ha kitörlik a kitöltő rekordját a „dolgozó" táblából (mert már nem dolgozik ott), vagy változik a dolgozó azonosítója (ezért nem szoktak elsődleges kulcsnak természetes adatot használni).

Az adatbázis-kezelők általában lehetőséget nyújtanak a hivatkozási integritás megőrzésére, ugyanúgy, ahogy a már korábban tárgyalt adatintegritás megőrzésére is. Ez egyrészt az adatbevitelnél, másrészt az adatmódosításoknál jelent korlátozásokat (megszorításokat).

Az egyedtípusok/táblák közötti kapcsolatok

Ahogy az egyedek, úgy az egyedek közötti kapcsolatok is tipizálhatóak. Például, ha valaki egy projektben részt vesz, akkor általában azt is el lehet mondani, hogy a dolgozók projektekben vesznek részt, tehát a dolgozó egyedtípus és a projekt egyedtípus, így a dolgozó tábla és a projekt tábla között is értelmezhető kapcsolat. Sőt elmondhatjuk, hogy a relációs adatbázisokban csak úgy lehet kapcsolat két egyed között, ha az egyedtípusaik között létrejön a kapcsolat: mint fentebb láttuk, az idegenkulcsok alkalmazásával. Az idegenkulcs, és amire hivatkozik, elsődleges kulcs párost *kapcsolómezők*nek is szokták nevezni, hiszen ez a pár teremti meg a kapcsolatot a két tábla között.

Az egyedtípusok, így a táblák között három fajta kapcsolatról beszélünk.

Az A egyedtípus **egy a többhöz** kapcsolatban áll a B egyedtípussal (jelölés: **1:N**), ha az A egyedei több egyedhez is kapcsolódhatnak a B-ből (lehet, hogy egyhez sem), azonban a B egyedei pontosan egy egyedhez kapcsolódnak A-ból. Az előző példákból merítve: egy dolgozó több számlának is lehet kiállítója (van, aki egynek sem), azonban egy számlának nem lehet több kiállítója, de egynek mindenképpen kell lenni.

Megvalósítás: a több oldalon, tehát a B táblájában alkalmazzuk az idegenkulcsot, mint már láttuk, ez a „számla" táblában a „kiállító" mező. Természetesen a kapcsolómező párja, az elsődleges kulcs az A táblában található („dolgozó").

Az A egyedtípus **egy az egyhez** kapcsolatban áll a B egyedtípussal (jelölés: **1:1**), ha az A egyedei legfeljebb egy egyedhez kapcsolódhatnak a B-ből (lehet, hogy egyhez sem), azonban a B egyedei pontosan egy egyedhez kapcsolódnak A-ból. Például egy dolgozó legfeljebb egy osztálynak lehet vezetője (van, aki egynek sem), és egy osztálynak sem lehet több vezetője, de egynek mindenképpen kell lenni.

Vegyük észre, hogy – bár a neve azt sugallja – ez a kapcsolat sem szimmetrikus, ami úgy fogható meg a legjobban, hogy a B táblájában nem szükséges ugyanannyi rekordnak lenni, mint az A-ban, hiszen A egyedeihez nem kötelező kapcsolódó egyednek lenni a B-ből (nincs minden dolgozónak saját vezetésű osztálya). A kisebb rekordszámú (B) táblában alkalmazzuk az idegenkulcsot, és a kapcsolómező-pár elsődleges kulcs oldalát abból a táblából vesszük, ahonnan mindenképpen kell egy rekordnak kapcsolódni, azaz az A táblájából. A példában az osztályok adatait tartalmazó táblában a „vezető" mező lesz az idegenkulcs, ami a megfelelő dolgozóazonosítókat hordozza. (Az „osztály" egyedtípus egyik tulajdonsága az, hogy ki a vezetője.)

A megvalósításban csak az az eltérés az egy a többhöz kapcsolattól, hogy az idegenkulcsra olyan megszorítást alkalmazunk, amely nem engedi meg azonos értékek ismétlődését. (Ne lehessen két osztálynak

ugyanaz a vezetője.)

Az A egyedtípus **több a többhöz** kapcsolatban áll a B egyedtípussal (jelölés: **M:N**), ha az A egyedei több egyedhez is kapcsolódhatnak a B-ből (lehet, hogy egyhez sem), és a B egyedei is több egyedhez kapcsolódnak A-ból (lehet, hogy egyhez sem). Az előző példákból merítve: egy dolgozó több projektben is részt vehet (van, aki egyben sem), és általában egy projektben több dolgozó vesz részt.

Megvalósítás: az A és B táblái közé iktatott, ún. kapcsolótábla létrehozásával, amely tartalmazza az A és a B táblára hivatkozó idegenkulcsokat. A példában lehet ez a „projektben részt vesz" tábla, mely a következő információkat tárolja: melyik dolgozó, melyik projektben, mettől, meddig, milyen szerepben vesz részt. Tehát a több a többhöz kapcsolatot két egy a többhöz kapcsolat oldja fel, az idegenkulcsok elhelyezkedéséből látható, hogy mindkettőben a több oldalon a kapcsolótábla áll.

2.4.2. A relációs modell előnyei

Redundanciamentesség

A relációs modellben az adatok ismételt tárolását, a redundanciát könnyen ki lehet küszöbölni az adathalmaz több táblára osztásával.

Például nem kell minden számlarekordban az ügyfél adatait tárolni, hiszen ugyanazon ügyfél számára több számlát is kiállíthatunk. Ha van egy olyan táblánk, ami kizárólag az ügyfeleink adatait tartalmazza ügyfél-azonosítóval is ellátva, akkor elég a számlarekordban a megfelelő ügyfélre csak az azonosítóját használva hivatkozni. Sőt más táblákban (panasz, levelezés) is hivatkozhatunk így az ügyfelekre anélkül, hogy adataikat újra meg újra tárolni kellene. Amint látjuk, szükség van minimális mértékű adatduplázásra, hiszen az ügyfél-azonosítókat mind az ügyfél (elsődleges kulcs), mind a ráhivatkozó táblában (idegenkulcs) tároljuk, de cserébe a sok egyéb adat (pl. név, cím, telefonszám, e-mail, számlaszám) ismételt tárolását, a redundanciát elkerüljük.

Teljes rugalmasság az adatbázis-kialakításban

Amint a kapcsolatok kialakításánál láttuk, bármelyik két rekord, azaz bármelyik két egyed között könnyen teremthető kapcsolat a relációs modellben: az egyedtípus csak egy tulajdonsággal, táblája egy mezővel bővül (idegenkulcs), mely a kapcsolódó egyed azonosítását végzi. Nincs szükség a kapcsolatok megvalósításához külön struktúra kiépítésére, mint a hierarchikus és a hálós modellek esetén.

Változatos és könnyű hozzáférés az adatokhoz

A relációs adatbázisok egyszerű műveleteket biztosítanak a felhasználók számára a tárolt adatok kezelésére és előhozására.

Skálázhatóság

A viszonylag könnyű létrehozáson és hozzáférésen felül a relációs adatbázis fontos előnye a könnyű bővítés. Szükség esetén az egyedek további tulajdonságai vagy akár egy új egyedtípus is integrálható a már működő adatbázisba a megfelelő táblákhoz új mezők felvételével vagy egy tábla létrehozásával és a meglévő táblákhoz való könnyű kapcsolatkialakításokkal. Ráadásul mindez anélkül kivitelezhető, hogy szükséges lenne az összes már működő eljárást módosítani.

A relációs adatbázisok **további fontos erősségei** között van a teljesítményük, a hatékonyságuk, az új hardvertechnológiák támogatása és a minden fajta adatigénynek megfelelő képességük. A felsorolt előnyök a célszerű logikai struktúrán kívül a fizikai adattárolás és a logikai szerkezet függetlenségének köszönhető.

2.4.3. Normalizálás

Amint az előzőekből láttuk, a relációs modell az adatokat táblákba foglalja. A táblák lehető legjobb kialakítása rendkívül fontos, mert ez alapfeltétele az adatbázis gyors, hatékony, zökkenőmentes működésé-

nek, könnyű kezelhetőségének és továbbfejleszthetőségének. Azt lehet mondani, hogy a relációs modell előnyei igazából a jól kialakított táblarendszer esetén érvényesülnek. Nagy jelentősége miatt módszert dolgoztak ki az adatok táblákba való szétosztására, amelyet *normalizálás*nak hívunk. Az eljárás lépcsőfokait, melyek az optimális szerkezethez való egyre közelebbi állapotokat írják le, pedig *normálformák*nak nevezzük.

Egy tábla **első normálformában (1NF)** van, ha nincsenek benne

- **többértékű mezők;**

Például a dolgozók által beszélt nyelvek felsorolása nem kerülhet egy mezőbe, de a lakcímének különböző részei (irányítószám, helység, utca-házszám) is önálló jelentéssel bíró adatok, tehát külön mezőkre kell szétválasztani.

- **ismétlődő mezők**

Például a dolgozó által beszélt több nyelv még úgysem kerülhet a dolgozó táblába, hogy ezeket külön mezőkben rögzítjük, mert nem tartalmazhat több nyelv mezőt a tábla. Ezt a problémát első lépésben úgy lehet feloldani, hogy ugyanannak a tulajdonságnak az értékeit ugyanabba a mezőbe, de külön rekordokba vesszük fel.

- **azonos rekordok, vagy ami ezzel egyenértékű: legyen a táblának elsődleges kulcsa.**

Például előfordulhat, hogy ugyanaz a vásárló egy napon két alkalommal ugyanabból az árucikkből azonos mennyiséget vesz, ám a két vásárlásrekordot meg tudjuk különböztetni a kiadott számla vagy nyugta számának felvételével (ez lehet az elsődleges kulcs). Vagy fordítva: ha nem is hozunk létre külön mezőt elsődleges kulcs céljából, de ha minden rekord különbözik, akkor szélső helyzetben az összes mező együttese is kinevezhető elsődleges kulcsnak.

Egy tábla **második normálformában (2NF)** van, ha

- első normálformában van;
- ha az elsődleges kulcsa összetett, akkor ennek valamely része nem határozza meg a tábla bármely más mezőjének értékét. (Egy mezőből álló elsődleges kulcs esetén ez a feltétel eleve teljesül.)

Például, ha a dolgozók azonosítójának egy része a dolgozó osztályára utal, akkor az osztályra vonatkozó bármilyen más adatot (például az osztály neve stb.) nem a dolgozó táblába, hanem egy erre a célra létrehozott külön táblába kell helyezni. Így alakul ki az osztály tábla, melynek elsődleges kulcsát az előbb említett kulcsrész értékeiből hozzuk létre. A tábla ketté válik, de összekapcsolja a létrejött két táblát a dolgozó tábla említett kulcsrésze (idegenkulcs szerepben) és az osztály tábla elsődleges kulcsa.

Egy tábla **harmadik normálformában (3NF)** van, ha

- második normálformában van;
- az elsődleges kulcstól különböző mező, vagy mezőkombináció nem határozza meg a tábla egyéb mezőinek értékét.

Például, ha a dolgozók végzettségéből és beosztásából egyértelműen következő mértékű pótlék jár, akkor fölösleges minden dolgozó adatainál a végzettségen és beosztáson kívül a pótlék összegét is feltüntetni, elég egy külön táblában meghatározni e három adat relációját. Az új tábla elsődleges kulcsát a végzettség és beosztás együtt alkotja, ami egyben a kapcsolatot is megteremti a dolgozó tábla ugyanezen mezői által (ott idegenkulcsként).

2.5. Lekérdező nyelvek

Tanulási célok

- Különbséget tesz a procedurális és nem procedurális lekérdező nyelvek között.
- Bemutatja a relációs algebra alapműveleteit, mint a szelekció, a projekció, az átnevezés, a Descartes-szorzat, az egyesítés, az illesztések és a halmazkülönbség.
- Bemutatja a strukturált lekérdező nyelv (SQL) összetevőit, mint az adatdefiníciós nyelv (DDL), az adatmanipulációs nyelv (DML) és az adatfelügyelő nyelv (DCL).
- Érti az SQL DDL parancsait, mint a CREATE, DROP, ALTER TABLE.
- Érti az SQL DCL parancsait, mint a GRANT és a REVOKE.

2.5.1. Procedurális és nem procedurális lekérdező nyelvek

Egy adatbázis-lekérdező nyelv segítségével a felhasználók interaktívan „faggathatják" az adatbázist, elemezhetik és módosíthatják (frissíthetik) az adatait jogosultságaiknak megfelelően. A lekérdező nyelvek két típusba sorolhatóak: *procedurális* és *nem procedurális lekérdező nyelvek*.

A **procedurális** lekérdező nyelvek nemcsak a célt határozzák meg, hanem a hogyant is. Például BASIC-ben vagy C-ben egy sor parancsot kell írni, amik a végrehajtandó eljárást alkotják. Más szóval meg kell határozni, lépésről lépésre hogyan jutunk el a válaszhoz. Normális esetben egy ilyen eljárás beolvas egy rekordot, feldolgozza azt, és a kapott eredmények alapján beolvassa a következő rekordot, amit hasonlóképpen feldolgoz, és így tovább, amíg a kívánt adatok össze nem gyűlnek.

A **nem procedurális** lekérdező nyelvek csak a célt határozzák meg, a hogyant nem. Például az SQL (Structured Query Language) és a relációs algebra. (Az SQL nyelvet egy adott célra alakították ki: a relációs adatbázisok adatainak lekérdezésére.) Itt a felhasználó egy utasítással megadja, hogy milyen adatokra van szükség, anélkül, hogy meghatározná, hogyan lehet ezekhez eljutni. Az adatbázis-kezelő lefordítja az utasítást, és az így kapott eljárás dolgozik a szükséges rekordhalmazon. Ez mentesíti a felhasználót az adatok belső szerkezetének, kinyerésének és átalakításának ismerete alól.

A nem procedurális nyelveket könnyebb megtanulni és használni, mint a procedurális nyelveket, itt a felhasználó kevesebb munkát végez, az adatbázis-kezelő pedig többet.

2.5.2. A relációs algebra alapműveletei

A relációs algebra a matematikának az a területe, amely a relációkon értelmezhető műveletekkel foglalkozik. Az itt használt terminológia alkalmazása az adatbázis-kezelésben matematikai alapra helyezi a relációs adatbázisok lekérdező műveleteit. Ne felejtsük, hogy a reláció, az oszlop és a sor fogalmak rendre megfelelnek a tábla, a mező és a rekord – az adatbázis-kezelésben megszokottabb – fogalmaknak.

A relációs algebra alapműveletei:

- A **szelekció** vagy **kiválasztás** a reláció sorai közül választja ki azokat, amelyekre szükségünk van. Például válogassuk ki az ezer forintnál olcsóbb árucikkek sorait: SELECT ár<1000 (cikk)
- A **projekció** vagy **vetítés** a reláció oszlopai közül választja ki azokat, amelyekre szükségünk van. Például a dolgozók nevére és telefonszámára lenne szükségünk: PROJECT név, telefon (dolgozó)
- Az **átnevezés** műveletet alkalmazzuk a relációk és az oszlopok átnevezésére.
- A **Descartes-szorzat** vagy más néven **keresztszorzat** két relációból egy újabb relációt hoz létre úgy, hogy a két reláció sorait egymásután illeszti az egyikből minden sort a másikból minden sorral kombinálva. Ennek megfelelően az eredmény sorainak száma a két reláció sorai számának szorzata lesz.
- Az **illesztés** vagy **összekapcsolás** két relációból egy újabb relációt hoz létre az előzőtől annyi eltéréssel, hogy nem minden sort kombinál minden sorral, hanem valamilyen feltétel alapján csak az egymásnak megfelelő sorokat kapcsolja össze.

- Az **unio** vagy **egyesítés** két relációból egy újabb relációt hoz létre, amely a mindkét reláció sorait tartalmazza eredeti formájukban. Természetesen ez csak úgy lehetséges, ha a két relációnak ugyanolyan oszlopai vannak.
- A **halmazkülönbség** két relációból egy újabb relációt hoz létre, amely az első reláció azon sorait tartalmazza, melyek nincsenek benne a második relációban. Ez szintén csak úgy lehetséges, ha a két relációnak ugyanolyan oszlopai vannak.

2.5.3. Az SQL felépítése

Az SQL (Structured Query Language, magyarul strukturált lekérdező nyelv) a relációs adatbázisok legelterjedtebb lekérdező nyelve, ezért minden valamirevaló relációs adatbázis-kezelő biztosítja felhasználói számár az SQL használatát. Így azt lehet mondani, hogy az SQL közös nyelv lett az összes relációs adatbázishoz, és – mivel ezeket alkalmazzák túlnyomó többségben – így majdnem minden adatbázishoz.

Az SQL négy alkotórészből tevődik össze:

- **adatdefiníciós nyelv** (**DDL**, azaz Data Definition Language) az adatbázis szerkezetének kialakítására és változtatására nyújt lehetőséget;
- **adatmanipulációs nyelv** (**DML**, azaz Data Manipulation Language) az adatbázisban tárolt adatokkal dolgozik;
- **adatfelügyelő nyelv** (**DCL**, azaz Data Control Language) az adatokhoz való hozzáférés szabályozására szolgál;
- **tranzakció-vezérlő utasítások** (Transaction Control Statements) a DML utasítások adatváltoztatásaihoz biztosítanak felügyeletet.

2.5.4. Az SQL DDL-parancsai

Az SQL DDL része az adatbázis kialakítására, azaz szerkezetének meghatározására és módosítására szolgál. Parancsai a **CREATE** (létrehozás), az **ALTER** (módosítás), a **RENAME** (átnevezés), és a **DROP** (törlés), amelyek a teljes adatbázisra, az adatbázis tábláira és az indexekre vonatkozhatnak. Ide szokták még sorolni a **TRUNCATE** (a tábla összes rekordját kitörli) és a **COMMENT** (megjegyzést tesz az adatszótárba) parancsot.

Példák:

- *CREATE DATABASE kolcsonzo;* – létrehozza a *kolcsonzo* adatbázist
- *CREATE TABLE cikk (leltszam Int, megnev Varchar(45), napiar Int, tipus Varchar(15));* – létrehozza a *cikk* táblát a *leltszam*, *megnev*, *napiar*, *tipus* mezőkkel.
- *ALTER TABLE cikk ADD beszerzes Date;* – *cikk* táblát bővíti a *beszerzes* mezővel.
- *ALTER TABLE cikk DROP beszerzes;* – a *beszerzes* mezőt törli a *cikk* táblából.
- *DROP TABLE cikk;* – törli a *cikk* táblát.
- *DROP DATABASE kolcsonzo;* – törli a *kolcsonzo* adatbázist

Ezekben és a további SQL példákban a nyelv kulcsszavainak kiemelésére csupa nagybetűs írásmódot használunk, az egyéb rögzített neveket (típus, függvény) nagy kezdőbetűvel, a példaspecifikus részeket pedig csupa kisbetűvel írjuk. Ez azonban nem nyelvi előírás, csak az átlátható szemléltetést szolgálja, hiszen az SQL nem kis/nagybetű érzékeny.

2.5.5. Az SQL DCL-parancsai

Az SQL DCL része az adatbázis objektumaihoz, így a bennük tárolt adatokhoz való hozzáférés szabályozására szolgál. Legfőképpen az adatbázis-adminisztrátor (DBA) alkalmazza, hiszen a megfelelő jogosultságok meghatározása az adatbázis-biztonság érdekében történik, ami az ő felelőssége.

A DCL utasítások az adatbázis szintjén szolgálják az adatbiztonságot. Az operációs rendszer szintjén is lehet szabályozni a felhasználói hozzáférést az adatbázishoz, például a felhasználói azonosítók és jelszavak alkalmazásával, azonban a DCL utasításokkal lehet a felhasználói jogosultságok és hatáskörök adományozását és visszavételét a legdifferenciáltabban szabályozni.

Alkalmazásuk az adatbázis használatbavétele előtt és mindakkor esedékes, amikor egy új felhasználó kerül a rendszerbe, amikor egy felhasználó jogosultságait korlátozni vagy bővíteni kell, amikor változik biztonságpolitika vagy egy különleges helyzet igényli, hogy valamelyik felhasználó végre tudjon hajtani egy bizonyos SQL utasítást. Például, ha egy felhasználó munkaköre megváltozik, a DBA visszavonja az előző munkakörhöz tartozó kiváltságait és az új szerephez szükséges felhatalmazásokkal látja el.

A **GRANT** parancs szolgál hozzáférési jogosultságok adományozása, és a **REVOKE** a GRANT paranccsal adott jogosultságok megszüntetése. A következő jogosultságokat lehet adni illetve visszavenni egy felhasználótól vagy szerepkörtől: CONNECT, INSERT, SELECT, UPDATE, DELETE, EXECUTE, USAGE.

Példák:

- *GRANT INSERT, DELETE ON cikk TO elado;* – az *elado* felhasználói csoport számára lehetővé teszi rekordok felvételét a *cikk* táblába és kitörlését a *cikk* táblából.
- *REVOKE INSERT, DELETE ON cikk TO elado;* – az *elado* felhasználói csoporttól elveszi a lehetőséget rekordok felvételére a *cikk* táblába és kitörlésére a *cikk* táblából.

2.6. SQL-lekérdezések

Tanulási célok

- Érti az SQL DML alap parancsait, mint az INSERT, DELETE, UPDATE, SELECT.
- Érti az SQL záradékokat, mint a WHERE, ORDER BY, GROUP BY.
- Vázolja a nézetek és a speciális SQL parancsok használatát, mint a COMMIT és a ROLLBACK.

2.6.1. Az SQL DML-parancsai

A DML, az adatmanipulációs nyelv a táblákban elhelyezkedő adatok kezelésére szolgál, mint az adatok kinyerése a **SELECT**, módosítása az **UPDATE**, új rekordok bevitele az **INSERT** és rekordok törlése a **DELETE** paranccsal. Vegyük észre a határozott különbséget a SELECT és a többi DML utasítás között: míg az előbbi csak megjelenít adatokat, az utóbbiak változást eredményeznek az adattartalomban.

Példák

Induljunk ki az adatdefiníciós parancsok példáinál létrehozott *cikk* táblából, és vegyük úgy, hogy már rögzítettünk bele három rekordot:

leltszam	megnev	napiar	tipus
120	motoros fűkasza	5000	kerti
298	sarokcsiszoló	3000	barkács
340	láncfűrész	6000	kerti

SELECT megnev, napiar FROM cikk; – a cikk táblából kigyűjti a megnev és a napiar mezők értékeit. Az eredmény:

megnev	napiar
motoros fűkasza	5000
sarokcsiszoló	3000
láncfűrész	6000

*UPDATE cikk SET napiar=napiar*1.1;* – megemeli az összes cikk kölcsönzési napi árát 10%-kal. A *cikk* tábla ezután:

leltszam	megnev	napiar	tipus
120	motoros fűkasza	5500	kerti
298	sarokcsiszoló	3300	barkács
340	láncfűrész	6600	kerti

INSERT INTO cikk VALUES (354, „fúrógép", 2000, „barkács"); – felvesz egy fúrógépet a cikkek közé. A tábla ezután:

leltszam	megnev	napiar	tipus
120	motoros fűkasza	5500	kerti
298	sarokcsiszoló	3300	barkács
340	láncfűrész	6600	kerti
354	fúrógép	2000	barkács

DELETE FROM cikk; – törli az összes cikket. A tábla ezután:

leltszam	megnev	napiar	tipus

Ritkán van szükség az összes rekord törlésére, ezért a következő szakaszban látni fogjuk, hogyan lehet szűkíteni a törlés és a többi adatmanipulációs parancs hatókörét.

2.6.2. SQL-záradékok

Az SQL utasításokban a parancsszót záradékok követik, melyek pontosítják, hogy mivel és hogyan történjen a parancs végrehajtása. A záradékok sorrendje kötött, és kulcsszavak határozzák meg a funkciójukat. Nem használ azonban külön kulcsszót a SELECT utasítás első záradéka: a mezőlista és az UPDATE első záradéka: a tábla megadása.

Kötelező záradékok az előbb említett kulcsszó-nélkülieken kívül a fönti példákból a **FROM**: a forrástábla, a **SET**: a frissítendő mezők és új értékük, az **INTO**: a céltábla megadása. A nem kötelező záradékok közül a leggyakoribbak a **WHERE**: a rekordok kiválasztására szolgáló feltételnek, az **ORDER BY**: az eredményrekordok rendezési kulcsának és a **GROUP BY**: a csoportosítás kulcsának meghatározására.

Példák

UPDATE cikk SET napiar=napiar+500 WHERE napiar < 2500; – a 2500 Ft-nál alacsonyabb napi árakat emeli 500-zal.

leltszam	megnev	napiar	tipus
120	motoros fűkasza	5500	kerti
298	sarokcsiszoló	3300	barkács
340	láncfűrész	6600	kerti
354	fúrógép	2500	barkács

SELECT megnev FROM cikk WHERE tipus = „kerti" ORDER BY megnev; – a kerti eszközök nevét mutatja névsorban.

megnev
láncfűrész
motoros fűkasza

SELECT tipus, Avg(napiar) AS atlagar FROM cikk GROUP BY tipus; – típusonként mutatja az átlagárat.

tipus	atlagar
barkács	2900
kerti	*6050*

DELETE FROM cikk WHERE megnev = „láncfűrész"; – a láncfűrész tönkrement, töröljük a cikkek közül.

leltszam	megnev	napiar	tipus
120	motoros fűkasza	5500	kerti
298	sarokcsiszoló	3300	barkács
354	fúrógép	2200	barkács

2.6.3. Nézetek és tranzakció-vezérlő utasítások

Nézet (view)

A nézetek tulajdonképpen elmentett választólekérdezések (SELECT utasítások), vagy nevezhetnénk ezeket virtuális tábláknak is, mert a táblák minden felhasználási módja rájuk is alkalmazható, azonban – a táblákkal ellentétben – adatot nem tárolnak. Jól kifejezi az elmondottakat a nézetet létrehozó parancs is, mint az alábbi példában láthatjuk.

Példa

A *CREATE TABLE barkacs AS SELECT leltszam, megnev, napiar FROM cikk WHERE tipus = barkács–;* utasítás végrehajtásával látszólag létrejön egy új tábla, amely csak a barkácscikkeket tartalmazza, azonban ennek adatai dinamikusan generálódnak: ha a cikk táblába felvesznek egy új barkácseszközt vagy kitörölnek egyet illetve átminősítenek – mondjuk – építőipari cikknek, akkor a *barkacs* virtuális tábla (nézet) is több vagy kevesebb sorral fog rendelkezni. Amikor a nézetet használják, akkor értékelődik ki a definíciójában lévő SELECT parancs minden alkalommal.

A nézetek használatának előnye, hogy az összetett, sokszor több táblát érintő adatkigyűjtések SELECT utasítását nem kell újra meg újra megírni, sőt – mivel, mint egy tábla, egy következő lekérdezés „alapanyaga" is lehet – további lekérdezések építhetők rá. Bár a példában mutatott *barkacs* nézet SELECT-je nem tekinthető bonyolult lekérdezésnek, azért a barkácsrészleg dolgozójának mégis segítség lehet, mert mindenféle kereséshez, kigyűjtéshez, összesítéshez használhatja azzal az előnnyel, hogy csak az őket érintő adatokat fogja szolgáltatni.

Másrészről a jogosultságok kiosztásánál is rendkívül hasznosak a nézetek, mert a GRANT utasításban használva lehetővé teszik, hogy ne csak teljes táblához adhassunk hozzáférést, hanem annak csak egy SELECT-tel kiválasztott részéhez is (mezők és rekordok között válogatva). Visszatérve a példánkhoz: ha az említett dolgozó nem a *cikk* táblához, hanem a *barkacs* nézethez kap hozzáférést, akkor ő csak a saját részlegénél használt cikkek adatait fogja látni. (Itt az elérhető rekordok halmazát szűkítjük.) Másik **Példa**: a különböző munkakörökben dolgozóknak más és más dolgozói adatokhoz kell és lehet hozzáférése, viszont az összes dolgozói adat egy táblában található. Itt az egyes nézetek, amelyekhez az egyes munkakörök jogosultságot kapnak, nem a rekordok, hanem a mezők közül fogják a szükségeseket kiválasztani (pl. némelyekben benne lesz a béreket tartalmazó mező, másokban nem).

Tranzakció-vezérlő utasítások

Az adatbázis-tranzakciók alapgondolata az, hogy egy csoport egymás utáni lépést úgy lehessen végrehajtani, hogy – ha szükséges – az összes lépést egyben vissza is lehessen vonni. Másrészt a tranzakció egy olyan elhatárolt területen kerüljön végrehajtásra, hogy közben az általa érintett adatokat más tranzakció ne módosíthassa.

Például egy pénzátutalási tranzakció lépései: az összeg levonása az egyik számláról és jóváírása a másikon.

Ha ezek közül bármelyik meghiúsul, a másik sem történhet meg, illetve ha megtörtént, akkor vissza kell vonni. Tehát egy eseményként kell kezelni ezeket a változtatásokat: vagy mindkettő megtörténik, vagy egyik sem.

A **COMMIT** parancs véglegesíti az őt megelőző utasítások által létrehozott változtatásokat. A **ROLLBACK** utasítással lehet visszavonni egy adott tranzakció hatásait.

2.7. Adatbázis-adminisztráció és –biztonság

Tanulási célok

- Bemutatja a legfontosabb adatbázis-adminisztrációs eljárásokat, mint a sémadefiníció, a tárolási struktúra és hozzáférési módok, a séma- és fizikai szerkezetmódosítás és az adathozzáférési meghatalmazás.
- Bemutatja a CIA (Confidentiality, Integrity, Availability = bizalmasság, sértetlenség, rendelkezésre állás) mozaikszóval jelzett biztonsági és integritási kérdéseket, mint az integritási megszorítások, a nem szándékos adatintegritás és konzisztencia veszteség és a szándékos (rosszindulatú) hozzáférés az adatbázishoz.
- Vázolja a különböző biztonságpolitikák példáit, mint a humán-, a fizikai, az operációs rendszer- és az adatbázis-védelem.
- Bemutat helyreállító módszereket különböző típusú meghibásodások, mint a logikai hibák, a rendszerhibák, a rendszerösszeomlás és a lemez-meghibásodás, esetén.

2.7.1. Az adatbázis-adminisztráció feladatai

Az adatbázis-adminisztráció célja, hogy biztosítsa a rendszer folyamatos elérhetőségét, adatainak integritását és adatainak biztonságát.

Az adatbázis-adminisztráció mindennapi tevékenységei:

- a rendszer figyelemmel kísérése a tényleges és lehetséges váratlan események szempontjából
- problémamegoldás
- adatbázis hangolása,
- kapacitás nyomon követése és jelentése,
- a rendelkezésre állás és a trendek elemzése és
- támogatást nyújt a katasztrófa elhárítási tervhez és a teszteléshez

Ezeket a tevékenységeket természetesen az adatbázis-adminisztrátor (DBA) végzi. Feladatainak további részletei:

- a rendszer kialakítására vonatkozó változtatási vagy továbbfejlesztési javaslatok felülvizsgálata és jóváhagyása;
- bármilyen módosításnak az adatbázison, a fájlok elrendezésén és elhelyezkedésén a felülvizsgálata és jóváhagyása;
- az adatbázis-kezelő rendszer tárhasználatával kapcsolatos jelentések tervezése és elemzése, a hely kiosztása és az adatbázis átszervezése, ha szükséges;
- kapacitástervezés és a rendszer jövőbeli erőforrásigényének a becslése;
- eljárások kidolgozása arra, hogy ellenőrizzék a felhasználók hozzáférését az adatokhoz, hogy biztosítsák az adatok integritását, hogy megakadályozzák a jogosulatlan hozzáférést és védjék az adatokat a nem szándékos és a szándékos helytelen használattól;
- eljárások fejlesztése a rendszer biztonsági mentésére és visszaállítására, az ügymenet folytonosságának biztosítására súlyos rendszerproblémák esetén.

2.7.2. CIA, mint bizalmasság, sértetlenség, rendelkezésre állás

Az információbiztonság kulcskérdése a cégadatok bizalmasságának, sértetlenségének és rendelkezésre állásának fenntartása. Csak az ezekkel jellemezhető információ használható a kereskedelmi tevékenységben. Ezen jellemzők bármelyikének csorbulása még a legnagyobb vállalatok fennmaradását is veszélyeztetheti.

Bizalmasság (Confidentiality)
A bizalmasság annak biztosítékát jelenti, hogy az információhoz csak a megfelelő személyek vagy szervezetek férnek hozzá. A bizalmasság sérülhet, ha az adatokat nem kezelik megfelelő módon, például kiszivárognak elmondás, nyomtatás, másolás, e-mail, létrehozott dokumentumok vagy más adatok útján. A bizalmasság megóvására alkalmazott egyik módszer az információk kategorizálása, ahol a kategóriákhoz tartozó megfelelő eljárásrend biztosítja a megfelelő védelmet. Például a dokumentumok minősítése lehet „bizalmas", „csak belső használatra" stb. Szintén a bizalmasságot szolgálják a jogosultsági rendszerek, amelyek felhasználónév és jelszó használatával egyedien azonosítják a felhasználókat, és így személyre lehet szabni az adateléréseket.

Sértetlenség (Integrity)
Az sértetlenség biztosítja, hogy az információ hiteles és teljes, ezáltal lehet számítani arra, hogy a céljának megfelelően szabatos. A sértetlenség kifejezést gyakran használjuk az Információbiztonság terén, mivel annak egyik alapindikátora. Az adatok sértetlensége nem csak annyi, hogy az adatok helyesek, hanem, hogy meg lehet bennük bízni és támaszkodni lehet rájuk. Például a másolat készítése (mondjuk a fájl elküldése e-mailben) minősített dokumentum esetén az információ bizalmasságán túl a sértetlenségét is veszélyezteti, mert a több előfordulás magában hordja módosulás kockázatát.

Rendelkezésre állás (Availability)
A rendelkezésre állás biztosítja, hogy az információ szolgáltatásáért, tárolásáért és feldolgozásáért felelős rendszer hozzáférhető legyen, amikor és akiknek szükséges. Korunkban elvárás az, hogy az adatok rendelkezésre álljanak a nap 24 órájában, a hét 7 napján, az év 365 napján azoknak, akiknek szükségük rájuk. Például ha valaki pénzt kíván felvenni egy ATM-ből, nem érdekli, milyen napszak van, egyszerűen csak ki akarja venni a pénzét. Az ehhez szükséges információknak mindig rendelkezésre kell állnia, hogy a rendszer számára lehetővé tegye a műveletet.

2.7.3. Biztonságpolitika

Fizikai szintű védelem

A biztonsági intézkedések általában a fizikai szinten kezdődnek: annak felügyelete, hogy csak az arra jogosultak jussanak oda az épületekhez, a szobákhoz, az iratokhoz és számítógépekhez. Ezt a célt szolgálják a beléptető rendszerek, amelyek eszközei lehetnek a beléptető kártyák és kártyaolvasók és a zárak az ajtókon, amelyek azt biztosítják, hogy csak azok juthassanak be, akik szükségességes, hogy ott legyenek.

Ehhez a szinthez tartozik még a papíralapú iratok tárolása és kezelése, többek között az archiválása. Általánosan bevett módszer, hogy az aktuális év irataihoz könnyen hozzáférnek a dolgozók, az előző évi iratokat biztonságosan elzárva tartják, a korábbiakat pedig egy külső kezelésű iratmegőrző központban.

Az információbiztonsági irányelvek dokumentálják, hogy a biztonsági mentések adathordozóit hogyan kezeljék és tárolják. Mivel ezek létfontosságúak a rendszer működése szempontjából, kezelésük – beleértve a biztonságos helyre és helyről való szállítást és a hosszú távú raktározást – szigorúan ellenőrzött.

Humán védelmi szint
Figyelmet kell szánni arra, hogy a dolgozók hogyan vigyáznak saját jelszavukra és a biztonságra. Oktatásra van szükség a biztonság fontosságáról, mert sok esetben az emberi tényező a leggyengébb láncszem a rendszerben. Gyakran látni fontos rendszerekhez szóló jelszavakat jegyzetlapokon, az asztalon vagy a billentyűzetre ragasztva. A dolgozók gyakran felügyelet nélkül hagyják az asztalukat, miközben egy kis szünetet tartanak, így az ott elhaladók számára elérhető az általuk használt rendszer. Számos cég felismeri a dolgozók oktatásának fontosságát arra a veszélyre, amit a munkaállomás lezáratlan elhagyása jelent, és büntetést szabnak ki a biztonsági szabályok megsértése esetén.

Szokásos biztonsági gyakorlat annak korlátozása, hogy az adott dolgozó milyen információkat nyomtathat, illetve, hogy hol és melyik nyomtatón nyomtathat. Ez a kérdés is az információbiztonsághoz tartozik. Példaként tekintsünk egy HR osztályt, amely rendszeresen végez lekérdezéseket a fizetések, az egészségügyi hozzájárulások, a nyugdíjjárulékok stb. megállapítására. Ez az információ nagyon érzékeny, és valószínű, hogy az ilyen jelentés nyomtatási példányai szigorúan limitáltak, csak a HR vezetők és kizárólag a helyi nyomtatókra nyomtathatják.

Operációs rendszer szintű védelem

Az IT csapat által érvényesített felügyelet biztosítja a szerverek és számítógépek védelmét a következő módokon:

- rendszeres szerver karbantartás és az operációs rendszer biztonsági frissítéseinek elvégzése;
- a szerverek, routerek és egyéb aktív hálózati eszközök zárt szekrényekben vagy erre a célra kialakított helyiségekben tartása;
- fizikai hozzáférés ezekhez csak az IT csapat tagjainak;
- a szerverekre rendszergazdai hozzáféréssel csak az arra felhatalmazottak tudnak bejelentkezni;
- a rendszergazdai jelszavak kezelésére szigorú előírások érvényesítése;
- a fontosabb szerverekhez való távoli hozzáférés szigorú korlátozása;
- az IT csapat javítócsomag-kezelési rendszere biztosítja minden szerver frissítését a legújabb operációs rendszer javításokkal, és minden munkaállomás megfelelő védelmét a rosszindulatú támadások ellen;
- vírusirtó szoftverek telepítése és rendszeres frissítése.

Hálózati szintű védelem

Hálózati biztonsági szabályozások a fent leírt módon biztosítják, hogy csak az arra jogosult felhasználók férjenek hozzá a fizikai rendszerhez, többek között a tűzfalakhoz és routerekhez. A rendszeres biztonsági auditok segítik érvényesíteni ezeket a szabályozásokat.

A behatolónak, aki megsérti egy cég hálózati vagy tárolási védelmét, a célja a következő három közül valamelyik: hozzájutni olyan információkhoz, amikhez nem szabadna hozzájutnia; megakadályozni, hogy a cég a saját adatait használhassa; károsítani vagy tönkretenni az adatokat. A cél lehet a hosszú távú károkozás is.

A biztonsági megfontolások két módon jelennek meg az adatvédelmi stratégiákban. Először is a biztonsági másolatok készítése rendkívül fontos akár a szándékos károkozásból, akár véletlen vagy rendszer-meghibásodásból adódó adatvesztés esetére. Másrészt a károkozással szembeni védelmi stratégia az adatok és információs javak megóvására. Bár a hálózati és szerverbiztonság elég fejlett, és az informatikai szakemberek megértették a fontosságát, a tárolási biztonság sokkal kevésbé kiforrott, akár a technológiát, akár a kialakult gyakorlatot nézzük.

Régebben az adatbázist a benne foglalt összes adattal egy szerveren tárolták. A mai tendenciák azonban a hálózati tárolás irányába mutatnak, ami új kockázatokat hoz az adatvédelembe. A hálózatalapú rendszer több hozzáférési pontot tartalmaz, ezért nagyon fontos minden ilyen pont beállítása úgy, hogy a hozzáférés csak a jogosult felhasználók számára legyen lehetséges. A hálózati tárolási környezetet sokkal összetettebb kezelni, mert az adateléréshez sokkal több eszköz és út a tendencia. Mivel a hálózati adattárolás egyik legfontosabb előnye a skálázhatóság, ezek a rendszerek általában gyorsan nőnek, és ezt nehéz lehet kezelni. Sok szervezet már ráébredt, hogy szükség van egy avatott adatbiztonsági szakember pozícióra a szervezeten belül.

Adatbázis szintű védelem

- **A biztonsági ellenőrzések beállítása** – Mielőtt bárki hozzáférhetne egy új adatbázishoz, az összes beépített biztonsági ellenőrzést be kell állítani. Minden felhasználónak kell kapni egy fiókot és jelszót, mely az adatbázisé, nem a felhasználó hálózati fiókja.
- **A patch-szint ellenőrzése** – Az adatbázis patch-szintű konfigurációját ellenőrizni kell az alapértelmezett beállítások gyenge pontjainak megállapítására. Teljes körű értékelést kell végezni az adatbázisban a meglévő rések kijavítására az adatok adatbázisba helyezése előtt.

- **Az adatbázis másolásának kizárása** – Jól bevált gyakorlat, hogy egyedül az adatbázis adminisztrátorának (DBA) van teljes hozzáférése az adatbázishoz. Ha az adatbázis egyszer le lett másolva, nincsenek ellenőrzés alatt az adatai, ezért meg kell tiltania adatbázis másolását, hiszen ez a belső fenyegetést jelent adatbázis-biztonságra.
- **A hozzáférés korlátozása** – Mint már említettük csak a DBA-nak lehet teljes hozzáférése az adatbázishoz, az összes többi felhasználónak a hozzáférését korlátozni kell a munkaköri szerepétől függően. A helyes gyakorlat az, hogy kialakítják a cég szerep- és a biztonságpolitikáját, ami meghatározza, hogy milyen hozzáférést igényelnek az egyes szerepek, így az adat hozzáférések összhangban lesznek a felhasználók munkaköri szerepével. Az adatbázis-adminisztrátor DCL utasításokkal megadja vagy visszavonja, a megfelelő hozzáférést minden felhasználónak az adatbázishoz.

A fent leírt biztonsági rendszerek és bevált gyakorlatok az adatbázisba épített biztonsági funkciókkal együtt jó biztonságpolitikát körvonalaznak egy cég számára.

2.7.4. Meghibásodások és helyreállítási módok

Egy cégnek rendelkeznie kell egy tervvel, amely leírja, hogyan álljon helyre egy hiba után a rendszer. Egy ilyen helyreállítási tervnek szorosan kell igazodni a cég üzletfolytonossági tervéhez, és részletesen le kell írnia az összes szükséges lépést ahhoz, hogy a rendszert helyre lehessen állítani olyan rövid idő alatt, ahogy csak lehetséges. A terv elkészítésénél figyelembe kell venni az összes lehetséges meghibásodásokat, de a részletek megtartása mellett a lehető legegyszerűbben leírva. Bele kell foglalni, hogy milyen adatokat szükséges lementeni, hol és hogyan kell tárolni ezeket, és hogyan lehet hozzájuk férni szükség esetén.

A meghibásodások skálája a logikai hibáktól a rendszer teljes hardverösszeomlásáig terjedhet.

Logikai hibák akkor fordulnak elő, amikor egy rossz vagy szabálytalan utasítás, formula a hibakeresés ellenére bennmarad, és csak ez általa okozott a rendszerösszeomlás derít rá fényt. Ilyenkor a hiba azonosítása elengedhetetlen a helyreállítás szempontjából.

A lemezhibák nem olyan gyakoriak, mint régebben voltak, és a RAID technológia fejlődésének köszönhetően nem okoznak problémát. A RAID (Redundant Array of Inexpensive Disks, azaz olcsó lemezek redundáns tömbje) technológia lehetővé teszi több kisebb és olcsóbb HDD kombinálását egy tömbbe úgy, hogy az együttes teljesítmény nagyobb, mint egyetlen nagy merevlemez esetén. A tömb egyetlen logikai tároló egységként, azaz meghajtóként jelenik meg az operációs rendszerben.

Totális rendszerhiba esetén a cég katasztrófa-helyreállítási tervének kell működésbe lépni. A legutóbbi biztonsági mentés tartalma rákerül egy másik szerverre, és a meghibásodott rendszert kiiktatják a hálózatból. Sok cég rendelkezik másodlagos backup szerverrel készenlétben tartva erre az esetre.

A virtuális rendszerek terén történt legújabb előrelépéseknek köszönhetően ma már lehetséges, hogy egy rendszer összeomlik, és visszaállítják anélkül, hogy a felhasználók észrevennék, hogy bármi történt volna. Az ilyen magas rendelkezésre állás ráfordítást igényel mind pénzben, mind magasan képzett szakemberben.

2.8. Adattárház, adatbányászat

Tanulási célok

- Bemutatja az adattárház (Data Warehousing) rendszer fogalmát és összetevőit.
- Meghatározza az adatbányászat fogalmát.
- Felismeri a DW rendszerek elvének alkalmazásait.

Adattárház

Az adattárház (DW, azaz Data Warehousing) rendszer elektronikusan tárolt adatok gyűjteménye, egy olyan adatbázis, melyek célja a vezetői jelentések és elemzések megalapozása. Az adattárház olyan adatokat tud szolgáltatni, amelyek összefoglaló képet nyújtanak a cég kondícióiról. Használata alapvető gyakorlat a vállalati jelentéskészítés, üzleti intelligencia és döntéstámogatás elősegítésére, és az alábbi tevékenységeket foglalja magába.

Az adattárház megtervezése és létrehozása
A kialakításnál ügyelni kell arra, hogy megfelelő alapot biztosítson a lekérdezések sokaságának, amely futni fog rajta. Ez az első lépés nagyon lényeges, mert a modell létrehozása és adatokkal való feltöltése után már nagyon nehéz visszatérni és módosítsa azt.

Adatgyűjtés
A cég adatainak áthelyezése a forrásrendszerekből a tárházba. Az adatok kivonatolása, átalakítása és eszközök betöltése történik ebben a folyamatban.

Megváltozott adatok befogása
A tárház adatait a rendszeresen frissíteni kell a tranzakciós rendszerből. Ezt bonyolítja, hogy nehezen lehet meghatározni, hogy a forrás mely rekordjai változtak meg a legutóbbi frissítés óta.

Adattisztítás
Az adattárházban lévő helytelen adat nemcsak haszontalan, hanem nagyon veszélyes. Az adattárház gondolata mögött a döntéshozatal megalapozása áll. Ha magas szintű döntés helytelen adatokon alapszik, akkor ez a cég számára súlyos következményekkel járhat, akár teljes összeomolással. Az adattisztítás egy bonyolult folyamat, amely a tárházba helyezés előtt ellenőrzi, és szükség esetén korrigálja az adatokat.

Adatösszesítés
Adattárházakat ki lehet alakítani úgy, hogy az adatokat a részletek szintjén tárolja, és úgy is, hogy egy bizonyos csoportosítási szinten, amely összesített adatokat tárol, vagy a kettő kombinációja is lehet. Az összesített adatok tárolásának előnye, hogy ezeket a műveleteket már nem kell elvégezni, tehát az összesítések kinyerése a tárházból jóval gyorsabb. Hátránya, hogy bizonyos részletinformációk, amelyekre szükség lehet, az összesítés során elvesznek.

Üzleti intelligencia
Az üzleti intelligencia rendszerek közé tartoznak a döntéstámogató rendszerek, a vezetői információs rendszerek és az online analitikus feldolgozó rendszerek (OLAP, azaz Online Analytical Processing System). Ezek jellemzően az alábbi eszközök egy részét vagy mindegyikét használják.

- Elemző eszközök, amelyek lehetővé teszik az adatok megvizsgálását több különböző nézőpontból.
- Lekérdező eszközök, amelyek lehetővé teszik SQL-lekérdezések kiadását és eredményhalmaz kinyerését a tárházból.
- Adatbányászati eszközöket, amelyek adatmintákat keresnek automatikusan az adatok között.
- Adatmegjelenítési eszközöket, amelyek grafikusan ábrázolva mutatják az adatokat, ideértve az összetett háromdimenziós adatképeket is.

2.8.1. Adatbányászat

Technikailag az adatbányászat az az eljárás, amely összefüggéseket vagy mintákat keres tucatnyi mező értékei között nagy relációs adatbázisokban. Azonban a kifejezést tágabb értelemben is használják: eljárás, amely különböző nézőpontokból elemez nagy mennyiségű adatot és összefoglalja hasznos információkká, olyan információkká, amelyek felhasználhatók a bevételek növelésére, a költségek csökkentésére vagy mindkettőre. Tehát a cél, megjósolni a jövőbeni trendeket és a vásárlói viselkedést, amely lehetővé teszi a vállalkozások számára az előrelátó, tudásalapú döntéseket.

Az adatbányászatot az OLAP (online analitikus feldolgozó rendszerek) különböző formáitól legegyszerűbben úgy lehet megkülönböztetni, hogy az OLAP csak olyan kérdésekre válaszol, amit fel tudunk tenni, míg az adatbányászattal olyan válaszokat is kaphatunk, amikhez nem feltétlenül tudunk kérdést megfogalmazni.

Például az eladási adatok összességét lehet promóciós törekvések szempontjából elemezni, hogy ismeretet szerezzünk a vásárlási szokásokról. Így egy kiskereskedő határozza meg, hogy mely elemek a leginkább fogékonyak a promóciós erőfeszítésekre. Az adatbányászat segítségével a kereskedő egyéni vásárlási történet alapján célzott promóciókat küldhet az ügyfél számára. Vagy egy részletes elemzést az összes üzletében történt vásárlásokról meg tudja mondani például, hogy milyen termékeket vásároltak együtt, hogy a hét melyik napján, és milyen típusú vásárló. Ezt az információt azután a marketing használja a termékelhelyezéshez és egyebekhez, amelyek serkentik az értékesítést.

2.8.2. Adattárházak alkalmazásai

Az adattárház (DW) segítségével megszerzett tudást a stratégiai tervezésben lehet használni és a fontos döntések meghozatalában, amelyek segítik a szervezet piaci sikerét a jövőben. Ha egy cég megérti a trendeket a DW használatának köszönhetően, fel tudja ismerni a múltban elkövetett hibákat, el tudja kerülni azokat a jövőben, és jobban tudja kezelni a jelenben.

Adattárházakat elsősorban érvényesítésre, taktikai jelentés készítésére és feltárásra használják.

Érvényesítés (validáció) az az eljárás, amelynek során a szervezet meggyőződik arról, hogy amit már feltételez, az tényleg igaz-e. Például már régóta sejtik a kereskedők, hogy különbség van a városlakók és a vidéki lakosok vásárlási szokásai között, majd az adatok elemzése igazolja ezt. Vagy egy cég a legjobb ügyfél meghatározásakor az összes adatot elemezve olyan választ kap, amely támogatja a korábbi álláspontját.

Taktikai jelentés készítéséről beszélünk, amikor egy cég taktikai célra akarja használni az adatokat. Például az eladással foglalkozó csapatnak szüksége van arra, hogy milyen promóció a legalkalmasabb egy adott országban vagy régióban, és adatelemzéssel készít beszámolót a korábbi vásárlásokról abban az országban.

A feltárás, az adatok keresése olyan ismeret vagy elgondolás számára, amivel nem rendelkeztünk előtte. Ez az, ahol a társítás, az osztályozás és a piaci alapú elemzés, mint adatbányászati technikák a legjobban számításba jöhetnek. Az adattárházak alkalmazásának ez a területe, amely a leghasznosabb lehet és a legnagyobb lehetőséggel rendelkezik.

2.8.3. Tesztkérdések

1.) Az alábbiak közül melyik tulajdonság szavatolja a tranzakciók végrehajtása után az adatok ellentmondás-mentességét és integritását?

i. Megbízhatóság.
ii. Elszámoltathatóság.
iii. Konzisztencia.
iv. Hatékonyság.

2.) Az alábbiak közül melyik NEM a tranzakciókra vonatkozó elvárás?

i. Tartósság.
ii. Elérhetőség.
iii. Atomiság.
iv. Izoláció.

3.) Az alábbiak közül melyeket kell figyelembe venni egy több-felhasználós rendszer kialakításakor?

A) Konkurens hozzáférés.
B) Redundancia.
C) Atomiság.
D) Adatintegritás.
i. Csak A.
ii. B és C.
iii. A, B és D.
iv. Mindegyik.

4.) Az alábbiak közül melyik NEM az adatbázis-kezelő rendszereknek a fájlkezelő rendszerekkel szembeni előnye?

i. Könnyebb a szoftverek újra-felhasználása.
ii. Könnyebb az adatbiztonság fenntartása.
iii. Könnyebb különböző típusú adatokat érintő lekérdezések futtatása.
iv. Könnyebb az adatintegritás fenntartása.

5.) Az alábbiak közül melyik szolgál a keresési és rendezési műveletek sebességének javítására?

i. Adatszótár.
ii. Adatállományok.
iii. Statisztikai adatok.
iv. Indexek.

6.) Az alábbiak közül mely területeken alkalmaznak adatbázis-kezelő rendszert?

A) Bankok.
B) Jegyirodák.
C) Áruházak.
D) Okmányiroda.
i. Csak A.
ii. B és D.
iii. A, B és D.
iv. Mindegyik.

7.) Az alábbiak közül melyik NEM az adatbázis-kezelő rendszerek használatának előnye?

i. Többszörös hozzáférés.
ii. Az inkonzisztencia fenntartása.
iii. Az adatok központosítása.
iv. Rugalmasság.

8.) Az alábbiak közül melyek tartoznak az adatbázis-kezelő rendszerek komponensei közé?
   A) Tranzakció-kezelő.
   B) Jelentésgenerátor.
   C) Adminisztrációs eszközök.
   D) Lekérdező nyelv.
   i. A és C.
   ii. B és D.
   iii. A, B és D.
   iv. Mindegyik.

9.) Az alábbiak közül melyik szerepkör felelős elsősorban az adatbázis biztonsági mentéséért és helyreállításáért?
   i. Adatbázis-adminisztrátor.
   ii. Adatbázis-felhasználó.
   iii. Adatbázis-tervező.
   iv. Adatbázis-programozó.

10.) Az alábbiak közül melyik adatabsztrakciós szint foglalkozik a tárolás struktúráival?
   i. A fogalmi szint.
   ii. A fizikai szint.
   iii. A logikai szint.
   iv. A felhasználói szint.

11.) Az alábbiak közül melyek tartoznak a rekordalapú logikai modellek csoportjába?
   A) Bináris modell.
   B) Hierarchikus modell.
   C) Objektumorientált modell.
   D) Relációs modell.
   i. A és C.
   ii. B és D.
   iii. A, B és D.
   iv. Mindegyik.

12.) Az alábbiak közül melyik a hálós modell megfelelő meghatározása?
   i. Rekordalapú adatmodell, melyben a rekordok közötti kapcsolatokat fagráfok írják le.
   ii. Rekordalapú adatmodell, melyben a rekordok közötti kapcsolatokat tetszőleges gráfok írják le.
   iii. Objektumalapú adatmodell, melyben az objektumok közötti kapcsolatokat fagráfok írják le.
   iv. Objektumalapú adatmodell, melyben az objektumok közötti kapcsolatokat tetszőleges gráfok írják le.

13.) Az alábbi állítások közül melyek helyesek az objektumalapú adatmodellekre?
   A) Az egyed-kapcsolat modellben az egyedeket attribútumok határozzák meg és kapcsolatok kapcsolják össze.
   B) Az egyed-kapcsolat modellben az objektumokat kapcsolatok írják le és egyedek határozzák meg.
   C) Az objektumorientált adatbázis az objektumorientált programozási rendszerek és az adatok hosszú idejű tárolásának kombinációja.
   D) Az objektumorientált adatbázis az objektumorientált programozási rendszerek és a relációs lekérdezőnyelv kombinációja.
   i. Csak B.
   ii. A és C.
   iii. A, B és C.
   iv. B és D.

14.) Az alábbi fogalmak közül melyik meghatározása a következő: „az idegenkulcsban csak olyan értékek fordulhatnak elő, amelyek szerepelnek a hivatkozott tábla elsődleges kulcsában”?

i. Elsődleges kulcs.
ii. Idegenkulcs.
iii. Hivatkozási integritás.
iv. Egyik sem.

15.) Az alábbi állítások közül melyek relációs modell jellemzői?

A) Mezőértékeinek a használata a rekordok összekapcsolására.
B) Fastruktúra használata a rekordok összekapcsolására.
C) A redundancia hatékony megvalósítása.
D) A redundancia kiküszöbölése.

i. A és C.
ii. A és D.
iii. B és C.
iv. B és D.

16.) A Részleg tábla a normalizálás melyik szintjén áll?
Részleg tábla: Részlegkód, Részlegnév, Helyszín, Menedzsernév, Menedzserazonosító, Telefon.

i. Normalizálatlan.
ii. Első normál forma.
iii. Második normál forma.
iv. Harmadik normál forma.

17.) Az alábbi állítások közül melyek hamisak?

A) A procedurális lekérdező nyelvekben meg kell határozni lépésről lépésre, hogyan juthatunk el a kívánt eredményhez.
B) A procedurális lekérdező nyelvekben elég meghatározni a kívánt eredményt.
C) A nem procedurális lekérdező nyelvekben meg kell határozni lépésről lépésre, hogyan juthatunk el a kívánt eredményhez.
D) A nem procedurális lekérdező nyelvekben elég meghatározni a kívánt eredményt.

i. A és C.
ii. A és D.
iii. B és C.
iv. B és D.

18.) Az alábbiak közül melyik relációs algebra műveletet használják a szükséges oszlopok kiválasztására?

i. Unió.
ii. Descartes-szorzat.
iii. Szelekció.
iv. Projekció.

19.) Az alábbiak közül melyek alkotórészei az SQL-nek?

A) DCL.
B) DDL.
C) DML.
D) Tranzakció-vezérlő utasítások.

i. Egyik sem.
ii. Csak D.
iii. A, B és C.
iv. Mindegyik.

20.) Az alábbi parancsszavak közül melyiket használja az SQL tábla törlésére?

i. DELETE.
ii. DROP.
iii. ERASE.
iv. REMOVE.

21.) Az alábbi szavak közül melyeket használja az SQL az adatokhoz való hozzáférés szabályozására?

A) AUTHORIZE.
B) GRANT.
C) PRIVILEGE.
D) REVOKE.

i. Csak A.
ii. Csak B.
iii. B és C.
iv. B és D.

2.8.4. Megoldások

1. iii.
2. ii.
3. iii.
4. i.
5. iv.
6. iv.
7. ii.
8. iv.
9. i.
10. ii.
11. ii.
12. ii.
13. ii.
14. iii.
15. ii.
16. iii.
17. iii.
18. iv.
19. iv.
20. ii.
21. iv.

3. rogramozás

3.1. Szoftvertervezési módszerek és technikák

Tanulási célok

- Felvázolja a különböző programtervezési módszerek, mint az objektumorientált (OO) tervezés, a „top-down" tervezés és a strukturált programozás főbb jellemzőit.
- Bemutatja az absztrakció problémamegoldási és szoftvertervezési módszerként való használatát.
- Felvázolja az örökölt rendszerek jellegzetes nehézségeit a szoftvertervezésben, mint a bonyolult szerkezet, a szegényes dokumentáció, az elavult szoftver/hardver és az üzletileg kulcsszerepű rendszer.
- Különbséget tesz a nyílt forráskódú és a tulajdonjoggal védett szoftver fejlesztése között.
- Felvázolja a tulajdonjoggal védett (proprietary), a nyílt forráskódú (open source), a szabad (free software) és az ingyenes (freeware) szoftverek nem összekeverendő licencezési követelményeit.

3.1.1. Programtervezési módszerek

Becslések szerint a szoftverfejlesztések költségeinek 70-80 százaléka az első kibocsátást követően keletkezik, mivel a hibák (bug-ok[2]) egy részére nem a tesztelési fázisban, hanem csak az éles használat megkezdésekor derül fény. Még akkor is előfordulhat újabb hiba, amikor már jó ideje használják a szoftvert. Javításuk a karbantartás része éppúgy, ahogy a frissítés, amikor a megváltozott környezethez (például a megváltozott gazdasági szabályokhoz) illesztik a programot. Frissítéskor nem kerülnek új funkciók a rendszerbe, mivel a továbbfejlesztés nem része a frissítésnek.

Az objektumorientált tervezés célja csökkenteni a szoftverrendszerek kidolgozásának és módosításainak költségeit. Az objektumorientált programszerkezet tükrözi a modellezett környezet szerkezetét. Egy forgalomszabályozó rendszerben például olyan struktúrákra van szükség, amelyek modellezik mindazt, amit a forgalomirányító a járművekről, az utakról, a hidakról, az alagutakról, a fizetőkapukról, stb. gondol.

Az OOP másik szigorú alapelve az, hogy a programstruktúrákban az objektumok elvárt viselkedését is reprezentálni kell, ami különösen előnyös, ha tudjuk, hogy azok a jövőben várhatóan változni fognak. Ez az alapelv biztosítja, hogy mindössze egy kódrészletet kelljen egy másikra cserélni ahhoz, hogy a program működését a változáshoz igazítsuk. Erre az elvre épül a komponens alapú programozás.

A módszer megkönnyíti a fejlesztők és a jövőbeli felhasználók közötti együttműködést: a felhasználó által megfogalmazott követelményhez egy programrészletet társíthatunk. Ha a követelmény megváltozik, pontosan tudjuk, hogy melyik az a programegység, amit a változás érint.

Az objektumorientált rendszer középpontjában nem a tevékenységek, hanem a tárgyak (dolgok) állnak: a szoftver tervezése során objektumokat és azok kölcsönhatásait modellezzük.

Az objektumorientált tervezés és programozás alapfogalmai

Osztály: hasonló tulajdonságokkal és viselkedéssel leírható objektumok halmaza. Az osztály leírja a modellezendő tárgy (fogalom) természetét, viselkedését, tulajdonságait; tulajdonságaikkal és az általuk végezhető műveletekkel határozza meg az objektumokat. Programozási szempontból az osztályeljárások és az adatdefiníciók egy csoportja, amelyből több azonos típusú objektumot deklarálhatunk. Az osztályban leírt eljárásokat az OO programozásban metódusnak nevezik.

2 A hibakeresést gyakran a „debug-golásnak" nevezik, a „bug" magyar jelentése poloska, a „debug" szó szerinti fordításban „poloskátlanítás".

Objektum: az objektum az osztály konkrét példánya, technikailag az OO tervezés alapegysége. A szoftver objektum az eljárások meghatározott csoportja, amelyek a valóságban létező objektumot (tárgyat/személyt/fogalmat) reprezentálják a programban, az objektumok aktuális állapotát meghatározó adatszerkezetekkel együtt.

Példány: a példány egy aktuális, egy adott osztályból futásidőben létrehozott objektum. A példány és az objektum szinonim fogalmak.

Adattag: az objektum tulajdonságát, állapotát reprezentáló változó. Minden objektum az osztályban deklarált, egységesen azonos nevű adattagokkal rendelkezik, de az adattagok értéke – az adott objektum állapotát meghatározó tulajdonságok - különbözhetnek.

Metódus: az objektummal (adattagjaival) műveletet végző eljárás vagy függvény. A metódusok határozzák meg az objektumok „viselkedését", vagyis azt, hogy az objektumok milyen feladatok elvégzésére vannak felkészítve.

Statikus adattag vagy metódus: az osztályhoz tartozik, minden objektumra egységesen jellemző. Statikus adattagot csak statikus metódus érhet el.

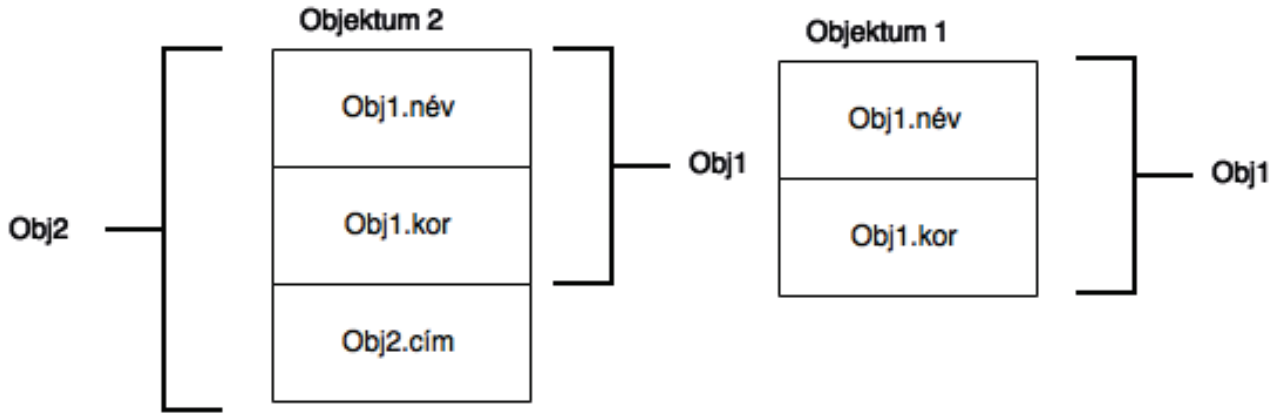
Üzenetküldés (method passing): az a folyamat, amely során egy objektum adatot küld egy másik objektumnak, vagy egy metódus végrehajtását kéri attól.

Metódus felüldefiniálás: egy osztályban meghatározott metódust az öröklés során az eredetitől eltérően definiálunk.

Öröklődés: az (ős-, alap-, szülő-) osztálynak van egy (utód-, gyerek-, származtatott-) alosztálya, ahol az alosztály jobban specializált, mint az eredeti osztály. A közös tulajdonságokat és metódusokat az ősben tároljuk, csak a speciális adattagok és metódusok kerülnek az utódba. Az utódosztályt ki lehet egészíteni újabb metódusokkal, vagy az ősben lévőt felüldefiniálni, hogy ugyanazon a néven, de máshogy működjön.

Polimorfizmus (többalakúság): az azonos őstől származtatott osztályok helyett használhatjuk az őst is, mintegy behelyettesítve az utódok helyére.

Az alábbi kép szemlélteti, hogyan valósul meg a memóriában az objektumok felépítése, Objektum 1 az ős és Objektum 2 az utód:



3.30. ábra Memóriakép, ami lehetővé teszi a polimorfizmust

Absztrakció: a problémának megfelelően modellező osztályokká egyszerűsíti a komplex valóságot, és egy adott probléma szempontjából szükséges tulajdonságokat, műveleteket fejt ki.

Az egységbezárás (encapsulation) és az adatrejtés (data hiding), igen gyakran összemosódó fogalmak. Az egységbezárás azt mondja ki, hogy az adatokat és a rajtuk végezhető műveletekért felelős eljárásokat egy egységként kell kezelni. Minden eljárást úgy kell meghatározni, hogy annak az objektumnak az adataival végezhessen műveleteket, amelyhez tartozik. Ehhez szorosan kapcsolódik az adatrejtés elve, ami pedig azt mondja, hogy az objektumoknak el kell rejteni az adataikat külvilág elől. Az adatokhoz mindig csak vala-

milyen ellenőrzött metódus formájában lehessen hozzáférni. Ez biztosítja, hogy a tervező tökéletesen meg tudja határozni az objektum belső működését, és ugyanakkor garantálni tudja az objektum viselkedését más, vele együttműködő objektumok tervezői számára.

Top-down tervezés

A top-down módszer a hangsúlyt a tervezésre, a rendszer teljes megértésére helyezi. A kódolást addig nem lehet megkezdeni, amíg a terv el nem éri a megfelelő szintű részletezettséget. A top-down módszer szerint a tervezést a teljes rendszer leírásával kezdjük, majd kisebb egységekre bontva haladunk a részletek kidolgozásával. A kis egységek elég részletesek és konkrétak lesznek ahhoz, hogy a kódolás, a programírás elkezdődhessen.

A tervezési módszerekről a lábjegyzetben szereplő linkeken lehet bővebben olvasni[3].

A módszer legfontosabb alapelve a lépésenkénti finomítás. A feladatot először átfogóan körvonalazzuk, majd részfeladatokra bontjuk, és a továbbiakban a részfeladatokat egyenként dolgozzuk ki. Az egyes részeket egymástól függetlenül (de egymáshoz illesztve) programozhatjuk, tesztelhetjük, javíthatjuk. Az egyes részeket addig kell finomítani, amíg elemi részfeladatokig nem jutunk.

A módszer néhány előnye:

- elválasztja az alacsonyabb szintű objektumot a magasabb szintű objektumoktól, ami moduláris felépítéshez vezet. A moduláris felépítés révén a fejlesztés a külső tényezőktől függetlenné (zárttá) tehető;
- kevesebb a hibalehetőség, mivel a programozó a modulokkal egyenként, a többi modultól függetlenül foglalkozik. Ez időmegtakarítást eredményez, és megkönnyíti a modulok összeépítését. [4]

Strukturált programozás

A strukturált programozás a procedurális programozás egy részhalmaza, amely megköveteli a program logikus szerkezeti felépítését, annak érdekében, hogy a kód hatékonyabb, könnyebben érthető és könnyen módosítható legyen.

A 60-as években és a 70-es évek elején, amikor a programozás egyre népszerűbbé vált, egyre-másra készültek a túlságosan komplex és egyre inkább áttekinthetetlen programok. A programok terjedelme nőtt, ami részben annak volt köszönhető, hogy jelentősen csökkent a lemezes háttértárak ára. De minél nagyobb volt a program, annál több volt a hibalehetőség. Gyakoriak voltak a programon belüli ugrások, emiatt a logikai felépítés nem volt átlátható, és a hibakeresés szinte lehetetlenné vált.

Ennek orvoslására fejlesztették ki a strukturált programozási módszereket. A strukturált programozás kulcsfontosságú alapelve a strukturált tervezés, amely szerint minden programot fel lehet építeni korlátozott számú szerkezeti elemből. Néhány szakértő állítása szerint mindössze három szerkezeti elemre van szükség: szekvenciára, elágazásra és iterációra.

A strukturált programozás gyakran alkalmazza a top-down tervezési módszert, amelyben a fejlesztő a teljes programot különálló szerkezeti egységekből építi fel. Annak, hogy a funkciójukban hasonló függvényeket egy különálló modulban kódoljuk, két előnye is van: a kód könnyen betölthető a memóriába, és a modul más programokban is használható. A modulokat csak akkor integrálják más modulokkal együtt a teljes programszerkezetbe, miután mindegyiket egyenként, különálló programegységként alaposan tesztelték. A program folyam egyszerű hierarchikus modell, amely különböző szerkezeteket tartalmaz.

A strukturált program három szerkezeti elemből épül fel[5]:

- elemi utasítások sorozata – azaz szekvencia;
- elágazás – azaz szelekció;
- ismétlés, ciklus – azaz iteráció.

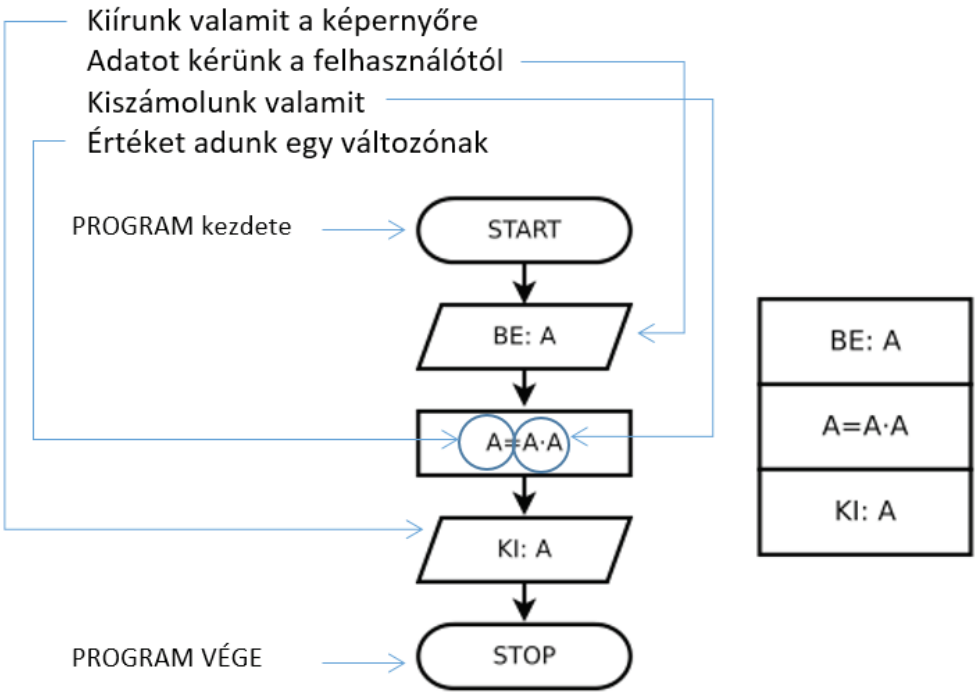
3 forrás: http://ntibi.web.elte.hu/programozas/elmelet/05_Prog_terv_modszerek.pdf
4 forrás: http://www.4kor.hu/mellekletek/AngsterErzsebet_Java1.pdf
5 képek és szövegrészek forrása: https://infoc.eet.bme.hu/ea02/

A program közvetlen vezérlésátadó (ugró) parancsot, tipikusan a *goto*, nem tartalmazhat. A *return, break, continue* megengedett, bár az utóbbi kettő megfelelő feltételekkel kiváltható. (A vezérlő szerkezetekről bővebben lásd a 3. fejezetet.)

Például, sokkal átláthatóbb a *while* utasítással megírt ciklus, amelyben a kilépési feltételt a ciklus elején megadjuk, mint ha a megszakítást a ciklus közepén egy *if* elágazással oldanánk meg.

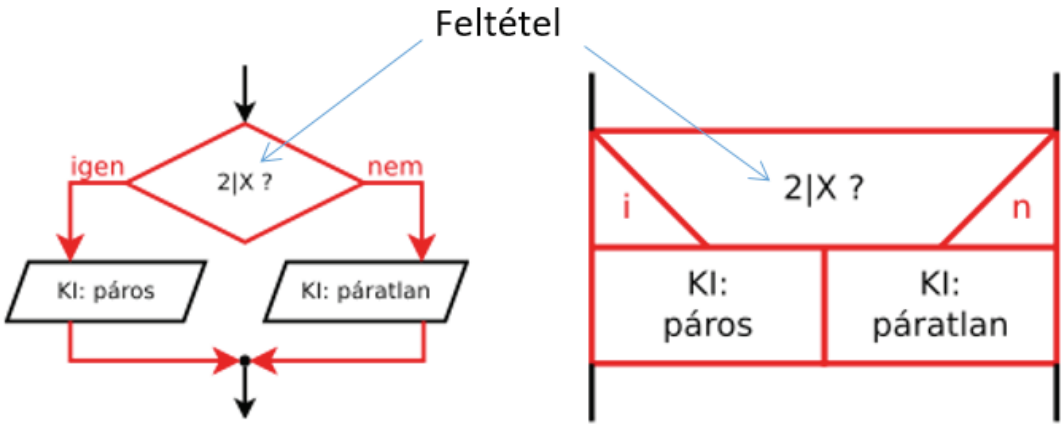
A következő ábrán kétféle jelölési mód látható: a baloldalon az úgynevezett folyamatábra, a jobboldalon pedig a struktogram. (A harmadik jelölési mód, a pszeudokód, illetve egy konkrét programnyelven való leírás a B.3 fejezetben található.)

Szekvencia: bármilyen utasítás.



3.31. ábra Szekvencia folyamatábra és struktogram (https://infoc.eet.bme.hu/ea02/)

Elágazás: egy bizonyos feltétel alapján más-más irányban halad tovább a program



3.32. ábra Elágazás folyamatábra és struktogram (https://infoc.eet.bme.hu/ea02/)

formai megjelenése:

```
if (feltétel)
        ha a feltétel igaz, ez a rész hajtódik végre
        elemi utasítások, szekvencia, szelekció
else
        ha a feltétel nem igaz, tehát hamis, ez a rész hajtódik végre
elemi utasítások, szekvencia, szelekció
```

Ciklusok fajtái: alapvetően két csoportba sorolhatjuk, annak alapján, hogy a ciklus:

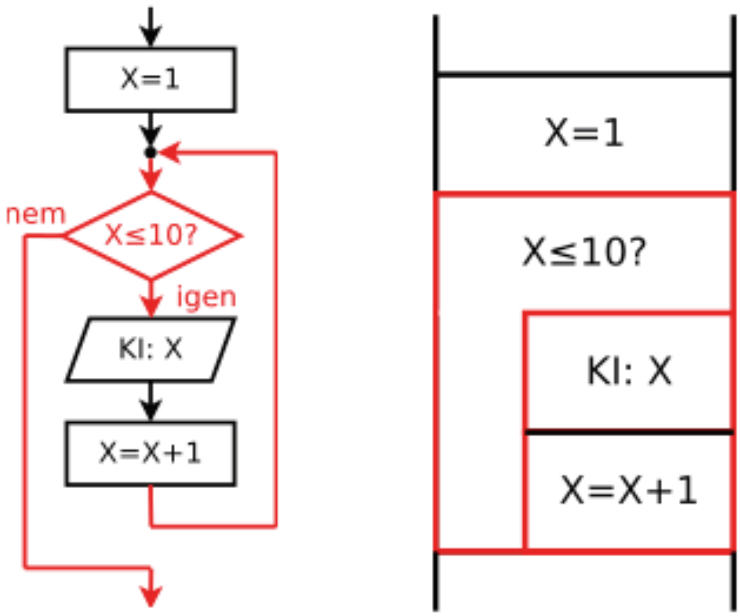
- elöltesztelő, vagy
- hátultesztelő

Az elöltesztelő ciklusok esetén – további felosztásként – megkülönböztetünk léptető ciklust is. Minden ciklus annyiszor hajtódik végre, ameddig a feltételében meghatározott kifejezés igaz marad. Többször felhasználni ugyanazt nagyon ritkán kell (pl.: kiírni ugyanazt a szöveget vagy számot), ezért a ciklusok belsejében találunk egy vagy több változót, ami folyamatosan új értéket vesz fel. Ha van ilyen, akkor ez a változó – a ciklusváltozó – vezérli a ciklust.

- Elöltesztelő: for, while, ebből a léptető ciklus a for
- Hátultesztelő: do/while

Formális jelöléssel:

```
for (értékadás; feltétel; léptetés) { utasítások }
while (feltétel) { utasítások }
do { utasítások } while (feltétel)
```



3.33. ábra Ciklusok folyamatábra és struktogram (https://infoc.eet.bme.hu/ea02/)

3.1.2. Absztrakció alkalmazása

Az absztrakció a szoftvertervezés egyik alapvető fogalma. Az absztrakció módszerével kiemeljük a jelenségek, tárgyak, fogalmak különböző variánsainak közös tulajdonságait.

Az absztrakció általánosítás, illetve annak eredménye; csökkenti a fogalom, vagy a megfigyelt jelenséghez kapcsolódó információ mennyiségét úgy, hogy csak azt veszi figyelembe, ami az adott probléma szempontjából lényeges. Például a „bőr futball-labdából” „labdát” absztrahálhatunk, ha a labda általános tulajdonságait és viselkedését (például kerek, pattog, gurul) emeljük ki. Hasonlóképpen, ha a boldogságérzést egy érzelmi állapottá absztraháljuk, akkor csökken az információ-tartalom, hiszen a konkrét érzelmi álla-

potra vonatkozó részletekkel nem foglalkozunk. A számítástechnikusok absztrakciót alkalmaznak a problémák megértésére és megoldására, majd a megoldást lefordítják egy speciális számítógépes nyelvre.

A vezérlés-absztrakció az a folyamat, amelyben a programozó új vezérlő szerkezeteket ír le és feltételeket határoz meg az utasítások végrehajtásának sorrendjére, függetlenül a konkrét implementációtól. A vezérlő szerkezetek gazdag választékával a programozó képes reprezentálni az algoritmusokban rejlő párhuzamosságot.

Mivel ugyanazon vezérlő szerkezetnek többféle implementációja létezhet, a programozó mindig kiválaszthatja a környezet (az adott hardver) adottságainak leginkább megfelelőt anélkül, hogy a forráskódot meg kellene változtatni. A legmegfelelőbb implementáció kiválasztása a program „tuningolásának" legegyszerűbb módszere.

Az adatabsztrakció módszere lehetővé teszi, hogy a programozó elrejtse azokat a részleteket, ahogyan az összetett adattípus létrejön az egyszerűbb adatobjektumokból.

Az adatabsztrakció elve szerint minden adattípushoz egyértelműen rögzíteni kell a vele végezhető műveletek halmazát, és az adott típusra a továbbiakban szigorúan csak ezeket alkalmazni.

A cél az, hogy a program összetett adatobjektumokat használjon, amelyek absztrakt adatokkal végeznek műveleteket. A programok csak annyi feltételezést tartalmazzanak az adatokról, amennyi az adott feladat elvégzéséhez feltétlenül szükséges. Ugyanakkor a konkrét adat reprezentáció definíciója legyen független az őt használó programoktól. A rendszer két részt összekapcsoló interfész szelektornak, illetve konstruktőrnek nevezett eljárásokból épül fel, amelyek létrehozzák az absztrakt adatok konkrét reprezentációit.

Az absztrakció lényegében a valós világ tulajdonságainak összegyűjtése, és leegyszerűsítése olyan mértékben, hogy az adott cél még éppen elérhető legyen.

Ha embereket akarunk egy olyan rendszerben modellezni, ahol csak az azonosításukra és a vizsgajegyük tárolására van szükség, akkor tulajdonságaik közül megtartjuk a nevüket, deklarálunk egy vizsgajegy változót, és nincs szükségünk további tulajdonságokra, mint például a nemük, hajszínük, súlyuk, magasságuk, stb.

3.1.3. Örökölt rendszerek hátrányai

Az elavult rendszerek elavult szoftvereket és elavult hardvert használnak. Azok a szervezetek, melyek kifutó modellekből álló rendszerekre támaszkodnak, nehéz választás elé kerülnek, amikor dönteniük kell a továbblépésről.

El kell dönteniük, hogy kiselejtezik a régi megoldást és módosítják az üzleti folyamatokat, vagy továbbra is fenntartják a régi rendszert.

A másik lehetőség a rendszer átalakítása az üzleti folyamatok újjászervezésével, a folyamatok jobb kezelhetősége érdekében. A kódvisszafejtés az a folyamat, amely a végrehajtható gépi kódból nyert információt további feldolgozásra alkalmas forrásnyelvű programmá konvertálja. Ily módon visszafejthető a rendszer architektúrája, melynek célja az, hogy a szoftver fő komponenseit azonosítsák a forráskódban.

Üzletileg kritikus rendszerek

Egy szervezet számára speciálisan kialakított szoftverrendszer rendkívül hosszú élettartalmú. Sok, a mai napig használtban lévő szoftverrendszert évekkel ezelőtt olyan technológiák felhasználásával fejlesztettek ki, melyek ma már elavultak.

Ezek a rendszerek üzleti szempontból kritikusak, és normális működésük elengedhetetlen az üzletben. Megváltoztatásuk folyamata egy új rendszerbe bonyolult, és gondos kezelést igényel.

Jelentős üzleti kockázata van annak, ha az elavult rendszert egyszerűen leselejtezik, és helyébe egy modern technológiával készült rendszert helyeznek.

Komplex struktúra

A legtöbb elavult rendszert különböző programok és közös adatok alkotják. Annak ellenére, hogy ezek a rendszerek jól meghatározott rendszerként kezdték meg az életüket, valószínű, hogy idővel már több átalakításon mentek át. Az ilyen elavult rendszereknek ritkán van teljes specifikációja.

Hosszú időn keresztül ezeket az elavult rendszereket különböző csapatok egészítették ki, így nem egységes a programozási stílus. Lehet, hogy a rendszer már elavult programozási nyelvet használ, és a fájlstruktúra összeegyeztethetetlen a jelenleg használttal. Az eredeti személyzet, akik ismerték a régi rendszert, valószínűleg már nem dolgoznak a szervezetnél, és lehet, hogy nehéz megszerezni a teljes ismeretet a különböző rendszerkapcsolati összetevőkkel együtt.

A legtöbb régi rendszert az objektum-orientált fejlesztés használata előtt tervezték meg, és ahelyett, hogy egymással kölcsönhatásban lévő objektumok csoportjaként szervezték volna ezeket a rendszereket, funkció-orientált tervezési stratégia segítségével alkották meg őket. Sokféle technika alakult ki arra, hogy helyet takarítsanak meg, vagy a sebességet növeljék, mivel a lemezterület költségei magasak voltak akkoriban, és ezek a folyamatok az egyszerűen megérthető rendszer rovására mentek.

Szegényes dokumentáció

A szoftver élettartalma során jelentős változásokon mehetett keresztül, amit esetleg nem dokumentáltak elég jól, és lehet, hogy volt néhány kisebb változás, amit egyáltalán nem dokumentáltak.

Az üzleti folyamatok függnek a régi rendszertől, és rendszerbe ágyazott üzleti szabályoktól, amelyeket nem dokumentáltak máshol.

Valószínű, hogy különböző csapatok dolgoztak más-más időpontokban a rendszer korszerűsítésén, ezért feltételezhető, hogy különböző módját választották a dokumentálásnak. Ezeket nehéz úgy összeegyeztetni, hogy teljes mértékben érthető legyen. Az szoftver eredeti szállítója - ha még működik is - nem tud olyan megbízható dokumentumot rendelkezésre bocsájtani, ami tükrözi a rendszer valós állapotát. Nem lehet tisztában azokkal a fejlesztésekkel, amiket a rendszeren végeztek az eredeti üzembe helyezés óta.

Elavult hardver

A rendszer szerkezete megsérülhet a sokéves kezelés során. Sok elavult rendszert nagyszámítógépes környezetbe építettek, melyeket ki kellett cserélni nagyobb teljesítményű és olcsóbb kliensszerver- vagy webalapú rendszerekre.

A hardver nem volt feltétlenül központosított, lehetett szétoszlatva különböző helyeken, melyek egymással összefüggtek, így a csere sokkal összetettebb.

Számos jelentős hardverfejlesztés fordult elő az elmúlt években, mind a lemeztechnológiában, mind az adatátvitelben, és a régi rendszerek nem kompatibilisek ezekkel az új eszközökkel.

3.1.4. A nyílt forráskódú és a tulajdonjoggal védett szoftver fejlesztése közötti különbség

A nyílt forráskódú szoftverfejlesztés koordinálatlan, de a lazán együttműködő programozók szabadon terjeszthető forráskódot használnak, és az Internetet használják kommunikációs infrastruktúrára. A nyílt forráskódú kifejezést nagyrészt a szabad szoftver kettős jellege miatt fogadták el.

A koncepció alapja maga a szabad szoftver filozófiája, amely alapvető jogként támogatja a szabadon hozzáférhető forráskódot. Azonban a nyílt forráskód kiterjeszti kissé ezt az ideológiát, hogy bemutasson több kereskedelmi megközelítést, mely magában foglal egy üzleti modellt és egy fejlesztési módszertant is. A nyílt forráskódú szoftvermozgalom egyik célja a szabad szoftver globális használatának elősegítése annak érdekében, hogy biztosítsa a szabadalmaztatott szoftverek teljes felszámolását. A nyílt forráskódú fejlesztés gyors evolúciós folyamatként írható le, amely nagyszabású szakértői értékelést használ fel. Ennek alapvető előfeltétele az, hogy lehetővé teszi a forráskód szabad módosítását, így ösztönözve az együttműködésen alapuló fejlesztést.

A nyílt forráskóddal fejlesztett szoftver általában a következő sémákat mutatja:

- A szoftvert fokozatosan lehet továbbfejleszteni,
- A felhasználókat társfejlesztőnek kell tekinteni, akiknek hozzáférésük van a szoftver forráskódjához. Ez arra ösztönzi a felhasználókat, hogy nyújtsanak be hibajavításokat, szoftverbővítményeket, dokumentációkat, stb.
- Minél több fejlesztő van, annál gyorsabban fejlődik a szoftver.
- **Korai kibocsátás:** a szoftver első verzióját a lehető leghamarabb szabaddá kell tenni, annak érdekében, hogy jó eséllyel találjanak korai társfejlesztőket.
- **Gyakori integráció:** az új kódokat olyan gyakran kell integrálni, amilyen gyakran csak lehet, hogy elkerülhető legyen a hibák nagy számának rögzítése a projekt életciklusa végén.
- Számos verzió kapcsán lehet hibát találni a szoftverben. Mikor a korai felhasználók megkapják a szoftver legújabb verzióját, az még nem hibamentes, így segítenek a hibák feltárásában és javításában.
- Nagyfokú modularitás: Minél több moduláris szoftver van, annál nagyobb tere van a párhuzamos fejlesztésnek.

Szabadalmaztatott szoftver: egy jogi személy tulajdona

Más személyek számára a szoftverhasználat licencszerződésekbe van foglalva.

A szoftver használatához kötött feltételek elég korlátozóak, és szokatlan, hogy a forráskódot is tartalmazza a szoftver. Forráskód nélkül lehetetlen módosítani vagy javítani a programon.

A legtöbb szabadalmaztatott szoftvercsomagot eladók terjesztik, akik a termék eladása mellett gyakran nyújtanak támogatást és karbantartást is a szoftverhez.

Sok szervezet számára ez a legfontosabb tényező, amikor dönteni kell egy új szoftver vásárlásáról.

3.1.5. Szoftverlicencezési típusok

A szoftverlicencezési eljárások a végfelhasználó és a szoftvergyártó között létrejött szerződésre irányulnak. Habár a szoftverlicenc lehet papíralapú megállapodás, leggyakrabban be van ágyazva magába a szoftverbe a telepítési folyamat részeként. Ha a felhasználó nem ért egyet a licenc feltételeivel, így akár egy kattintással jelezheti azt. Ez megszakítja a telepítési folyamatot. A legtöbb esetben a felhasználók az egyetértést választják, függetlenül attól, hogy elolvasták-e az engedélyt, vagy sem.

Szabadalmaztatott szoftver
A szabadalmaztatott szoftver a felhasználónak csak egy konkrét célra ad engedélyt, és a szoftverhasználathoz kapcsolódó feltételeket egyértelműen kell megadni. A szoftver tulajdonjoga továbbra is a kiadóé. Ez a licenc tartalmaz egy részletes listát azokról a tevékenységekről, amik korlátozva vannak, mint például: a szoftver egyidejű használata több felhasználó által, összehasonlítások és teljesítménytesztek közzététele. A forráskódot soha nem adják ki a szoftverhez. Két gyakori típusa van a saját fejlesztésű szoftver licenceknek.

Végfelhasználói licencszerződés
Ez a leggyakoribb engedélytípus; az EULA (End User License Agreement, azaz végfelhasználói lincencszerződés) előírásai rendelkeznek arról, hogyan lehet egy szoftverrészt használni egy szervezeten belül. Néhány szoftvert telepíthetnek több gépre, ha csak egy fut egy időben. Ugyanakkor más szoftvereket csak egy gépen lehet betölteni. Általában nem lehet egyszerre több példányban használni a szoftvert azonos időben az EULA engedélye alatt. Ha egy szoftver, pl. operációs rendszer úgymond dobozos, akkor az gyakorlatilag korlátozás nélkül tovább értékesíthető, ha OEM (rendszerépítő), akkor a feltelepített gépre érvényes csak. Az OEM ára jóval kedvezőbb.

Mennyiségi licencszerződés
A mennyiségi licenc megállapodások általában lehetőséget biztosítanak a szervezetek számára, hogy több engedélyt vásároljanak. A szoftverszállítóknak különféle programjai lehetnek a szervezet típusától függően, lehet oktatási, jótékonysági vagy társasági célú.

Például sokan használják közülük napi rendszerességgel a Microsoft Office programcsomagot. Azonban az engedély feltételei teljesen mások lehetnek ezekre. A nagy szervezet mennyiségi licenccel lehetővé teszi több száz felhasználó hozzáférését különböző helyeken. Ha egy felhasználó egy kis irodában vagy otthon dolgozik, más licenc típust fog használni. Hasonlóképpen kormányzati hivatalok és az oktatási intézmények másmilyen engedélyt szerezhetnek a személyzet vagy a diákok számától és típusától függően. Ha a diák nappali tagozatos, akkor lehetséges, hogy diákkedvezménnyel olcsóbban kap engedélyt a szoftverhez.

Nyílt forráskódú szoftver

A nyílt forráskódú szoftvereknél a szoftver egy adott példányának tulajdonjoga nem marad a szoftver gyártójánál. Ehelyett a másolat tulajdonjoga a végfelhasználóé lesz. Ennek eredményeként a végfelhasználó megkapja a szerzői jog által biztosított összes jogot a másolat tulajdonosaként. A szoftver tartalmazza a forráskódot, és az engedély lehetővé teszi a kód módosítását.

A Copyleft szoftver kifejezés leírja a szabad szoftvert, amelynek terjesztési feltételei nem engednek a terjesztőknek semmiféle további korlátozást, amikor terjeszti a szoftvert vagy módosítja azt. Ez azt jelenti, hogy a szoftver minden példányának szabad szoftvernek kell maradnia, még akkor is, ha időközben módosították. Mindig megosztják a módosítások forráskódját, és általában nagyon kevés korlátozás van a használat és a forgalmazás terén.

A **copyleft** kifejezés angol szójáték, a copyright megfordításának eredménye. A copyleft lényege, hogy a jog adta eszközöket nem az adott szellemi termék terjesztésének gátlására, hanem a megkötések kiküszöbölésére használják fel, így garantálva a felhasználás szabadságát a módosított változatokra nézve is.[6]

Szimbóluma a copyrightot jelképező © megfordítása, aminek azonban a jog nem tulajdonít jelentést.

3.34. ábra Copyleft a Wikipédia oldaláról

Szabad szoftver

Sokfajta szabad szoftver létezik, különböző célokra. A mobil eszközök elterjedésével, mint például az iPhone, óriási növekedés következett be az ezekre az eszközökre szánt applikációk fejlesztésének számában. A legtöbb ilyen alkalmazás ingyen letölthető és telepíthető. Példa erre az Xe-Currency converter, ami lehetővé teszi a felhasználó számára, hogy ellenőrizze például, az euro-dollár árfolyamot, amikor rendelkezik internet-hozzáféréssel.

Az ingyenes szoftver, mint azt a neve is mutatja, ingyenes, habár továbbra is szerzői jog védi, és a módosítása, valamint a használata korlátozott. Mivel a legtöbb ingyenes szoftver szerzője bízik szoftverének lehetséges nagyközönségében, a terjesztésre vonatkozó szabályok általában lazábbak, mint a szabadalmaztatott szoftvernél, de a szerzők még mindig nem akarják engedni a szoftverük módosítását.

Sokfajta ingyenes szoftver létezik, példa rá a Darkwave Studio, amely egy nagy teljesítményű virtuális stúdió, kiváló minőségű digitális zene létrehozására.

6 forrás: https://hu.wikipedia.org/wiki/Copyleft

3.2. Adatstruktúrák és algoritmusok

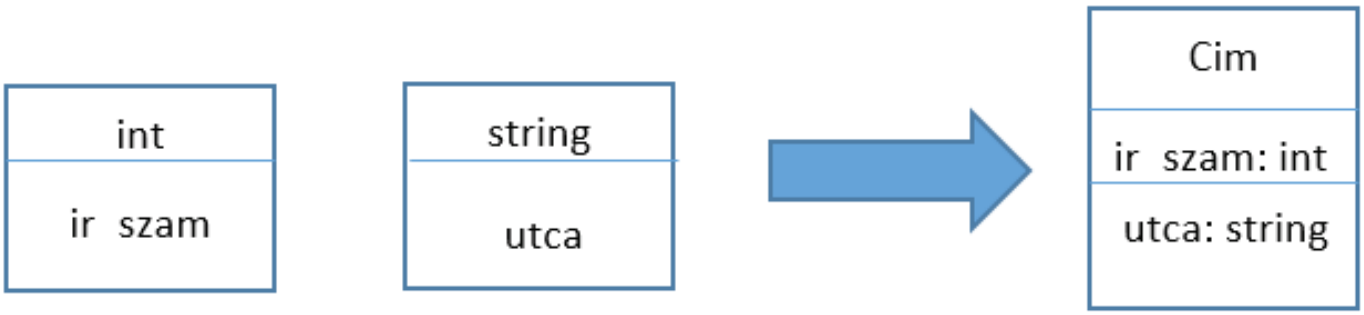
Tanulási célok

- Bemutatja a strukturált és strukturálatlan adattípusokat, és azonosítja a különböző adatstruktúrákat, mint a rekordok, a tömbök és a láncolt listák.
- Kiértékeli a tipikus keresési és rendezési algoritmusok és a különböző adatstruktúrák közötti illeszkedést.

3.2.1. A strukturált és strukturálatlan adattípusok és különböző adatstruktúrák

A strukturált vagy összetett adattípus a felhasználó által definiált adattípus, amelyeket az egyszerű (és/vagy összetett) típusokból képezzük valamilyen konstrukciós elvvel[7].

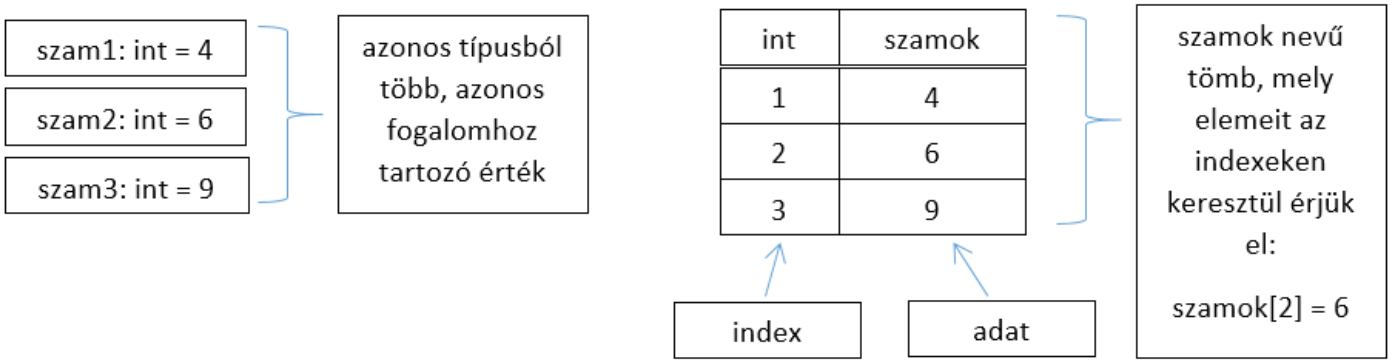
Nézzük meg a Cím strukturált típusunkat, mely állhat egész számokat tartalmazó irányítószámból, szöveges elemi típusból, ami az utca nevét mutatja, és így tovább.



Strukturálatlan adattípus olyan összetett adattípus (pl. szöveg, kép, zene, videó), ahol az elemek egy elemgyűjteménybe kerülnek, és innen lekérdezhetők. Az összetett szerkezetben nincs összefüggés az elemek között, attól a ténytől eltekintve, hogy ugyanabban a struktúrában vannak.

Egy tömb a meghatározás szerint olyan adatszerkezet, amely azonos típusokból tartalmaz valamennyit. Akár tömböt is tartalmazhat. A tömb lehet dinamikus, asszociatív, egy vagy több dimenziós.

Tipp **Egy tömböt úgy lehet elképzelni, mint index által hivatkozott értékek egy egyszerű listáját.**



Az ábrán az indexelés 1-3 ig terjed, de a programnyelvek többnyire 0-val kezdik az indexelést!

7 forrás: http://www.uni-miskolc.hu/~matha/Progelm_07ea.pdf

Egy C nyelvű programban egészek tárolására alkalmas 3 elemű tömb deklarálása és inicializálása a következő:

```
int szamok[3];
szamok[0] = 4;
szamok[1] = 6;
szamok[2] = 9;
```

deklarálás

inicializálás

egy sorban a deklarálás és inicializálás:

```
int szamok[]={4,6,9};
```

Csak inicializálásra, azaz kezdeti értékadásra használható!

Léteznek az egyszerű, beépített típusok minden nyelvben, ezek számok, karakterek. Nyelvfüggően, de megvalósításra kerültek logikai-, szöveges- vagy akár dátumtípusok is. A típus egyértelművé teszi, hogy milyen műveletek értelmezhetőek rajtuk. Különböző módokon, de további felosztásra kerülnek a számok, előjel, egész, valós és ábrázolási tartomány formájában.

Ha több azonos típust akarunk tárolni, akkor adódik, hogy tömböt kell használjunk. Ha pl. neveket akarunk eltárolni, akkor ezek egyenként szöveges alaptípusúak, tehát a tömbünk is szöveges típusú lesz, csak éppen több értéket tárol.

Ha szükséges a nevek mellé mondjuk a születési év, akkor dátumtípusra is szükségünk lesz. Ha a nyelv nem támogatja a dátumtípust, akkor számként tároljuk, vagy létrehozhatunk saját, összetett típust is. A lényeg, hogy az eddigi szöveges típustól eltérő típusra lesz most szükségünk. Megtehetjük, hogy létrehozunk egy új tömböt, ami több dátumtípusú adatot fog tartalmazni. Ezzel az a baj, hogy több tömböt kell kezelnünk, ezekben igazából egymástól függetlenül tároltuk a nevet és a hozzátartozó születési évet.

A jó megoldás, hogy létrehozunk egy saját struktúrát, ami különböző típusokat képes tárolni. A C nyelv ezt „struct”-nak, azaz struktúrának, a C++, C#, Java „class”-nak azaz osztálynak, a Pascal „record”-nak, azaz rekordnak hívja. A lényeg a tömbhöz képest, hogy míg a tömb csak azonos típusokat tud tárolni, addig a rekord (ez a kifejezés maradt meg, nyelvenként máshogy kell megvalósítani) bármilyen típusokat tárol.

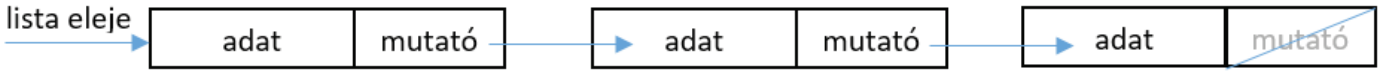
A rekord azonban csak 1 adatsort tárol, mint egy táblázat 1 sora. Ha több ilyen adatsort szeretnénk, mint a teljes táblázat, akkor a rekordot, mint típust, tömbként kell létrehozzuk.

egy rekord, ami 1 címet tárol	Cím típusú tömb, ami cimek néven több címet tárol
Cím{ir_szam: int, utca: string }	Cim | cimek 1 | Cím{ir_szam: int, utca: string } 2 | Cím{ir_szam: int, utca: string } 3 | Cím{ir_szam: int, utca: string } elérése: cimek[2].ir_szam = 1025

A tömbökben tárolt értékek, akár elemi típust, akár rekordot tartalmaznak, a memóriában egymás után helyezkednek el. Ez azt is jelenti, hogy a program futása végéig fix méretűek. Át kell írni a kódot, ha változtatni kell a méreten. Ha futási időben (a program működése közben) kell változzon a mérete, akkor az alábbi megoldásokból választhatunk:

Dinamikus tömb: továbbra is fix méretet kell allokálni (használatba venni), de bizonyos körülmények hatására nagyobb vagy kisebb területet foglalunk le, és ebbe átmásoljuk az eredeti tömb adatait.

Láncolt listák: A memóriában nem egymás után helyezkednek el az adatok, hanem „össze vannak láncolva”. Minden adat tartalmaz egy mutatót, ami megmondja, hogy a következő adat melyik memóriacímen található. Ha egyirányú a láncolt lista, akkor csak előre haladhatunk, ha kétirányú, akkor egy elemről az előző és következő elem is elérhető.



Objektumorientált környezetben a lista, ami tartalmaz minden adatot, egy osztály. Minden adat, ami a listában van, egy csomópont, szintén egy külön osztály alkotja. Tehát a Lista osztály példánya tartalmaz Adat osztályból példányokat. Minden Adat úgy épül fel, hogy tartalmazza a megjelenítendő tényleges adatot, ez lehet egyszerű típus, és tartalmazza a következő Adat típusú elem memóriacímét is. Ennek a típusa C++ nyelvben mutató, egyébként referenciának hívjuk.

A verem adatszerkezet megvalósítható statikus vagy dinamikus tömbből, vagy bármilyen listából. A lényege, hogy a **legutoljára** beletett elem érhető el elsőre. Az elemek elérésének a sorrendje számít. További elnevezései: stack, LIFO (Last In First Out)

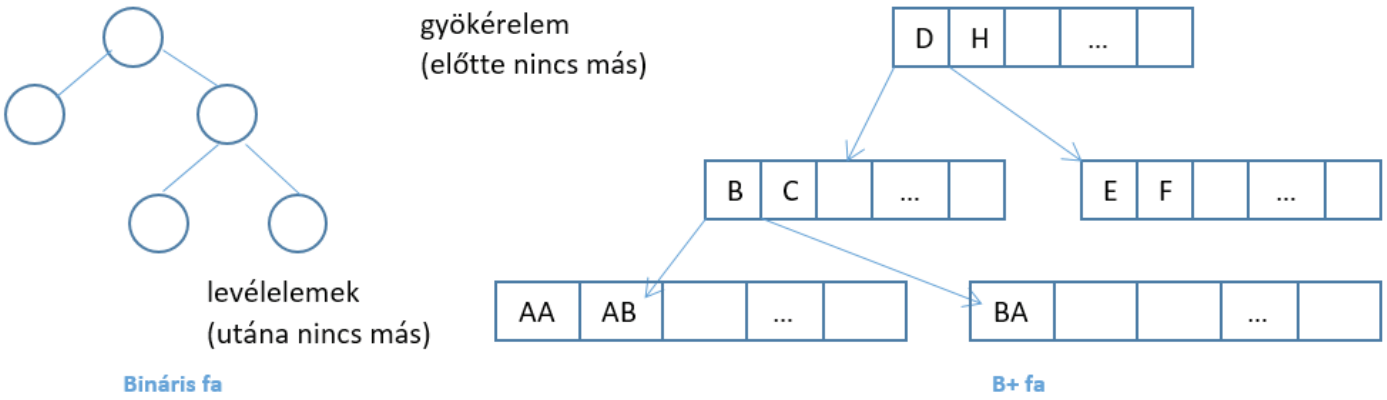
Az egyik módja, ahogy el tudjuk képzelni ennek a gondolatnak a megvalósítását, ha egymásra halmozott tányérokra gondolunk. Ami utolsónak kerül a verembe, az lesz az első, amit levesznek.

A sor adatszerkezet is megvalósítható statikus vagy dinamikus tömbből, vagy bármilyen listából. A lényege, hogy a **legelsőnek** beletett elem érhető el elsőre. Szintén az elemek elérésének a sorrendje számít. További elnevezései: queue, FIFO (First In First Out)

Léteznek még különböző adatszerkezetek, speciális feladatokra. A számítástechnikában kiemelt szerepe van a bináris fáknak. Beszélhetünk bináris keresőfáról[8] mely tárolási elvét a manapság elterjedt a B-fáknál is viszont lehet látni. Itt is kiemelünk egy fát, ami adatbázis kialakításoknál gyakori, a B+ fát[9].

Minden adatszerkezetre megfelelő műveleteket kell biztosítani. Közösnek mondhatjuk a beillesztést, törlést, módosítást. A keresés már erősen adatszerkezet-függő, ráadásul különböző sorrendben is bejárhatjuk egy fa csomópontjait. Erről a következő oldalon olvashatunk.

A fák csomópontokból épülnek fel, némelyik csomópontnak kitüntetett neve van:

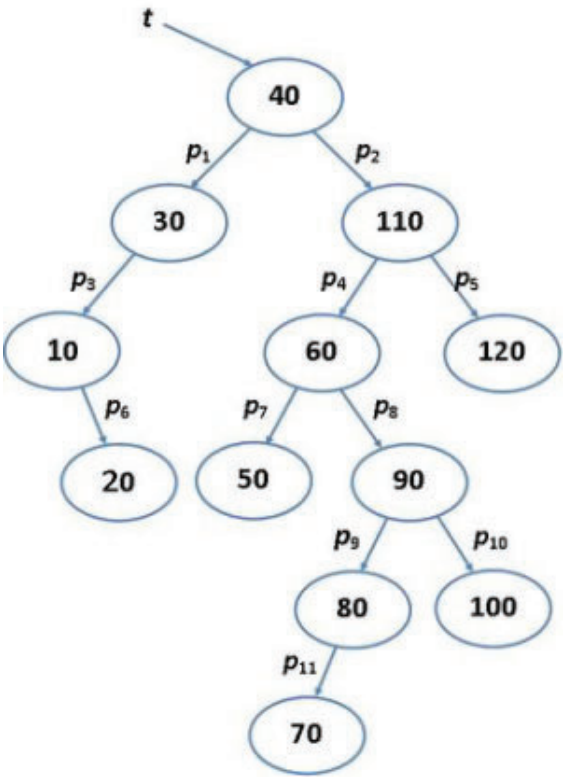


A bináris fa 0 vagy 2 elemmel követheti a megelőző csomópontot.

A B+ fa esetén a felül elhelyezkedő listából újabb listákra történik hivatkozás

A bináris keresőfa lényege, hogy az első elem lesz a gyökér, a továbbiakban az ennél kisebb balra, a nagyobb jobbra kerül. A 40, 110, 30, 60, 90, 10, 50, 20, 120, 80, 70, 100 sorozat tárolása bináris keresőfában:

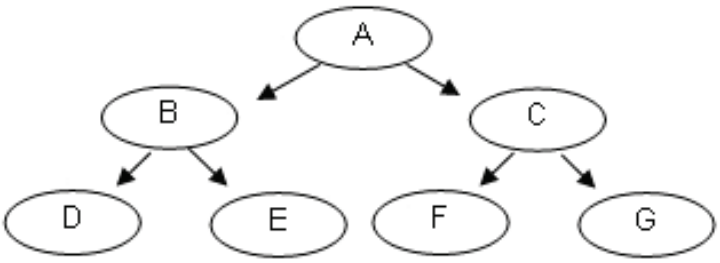
8 forrás: http://tamop412.elte.hu/tananyagok/algoritmusok/lecke12_lap1.html
9 forrás: http://www.kobakbt.hu/jegyzet/AdatbazisElmelet/ora1_index.html



3.35. ábra Bináris keresőfa

A fák bejárása különböző módokon történhet. Ez alatt azt értjük, hogy más-más sorrendben vesszük az összes csomópontot[10]:

Preorder bejárás: ha a fa üres, akkor vége az eljárásnak. Ha nem, akkor dolgozzuk fel a gyökérelemet, majd preorder eljárással járjuk be a bal oldali részfát, majd a jobboldali részfát.

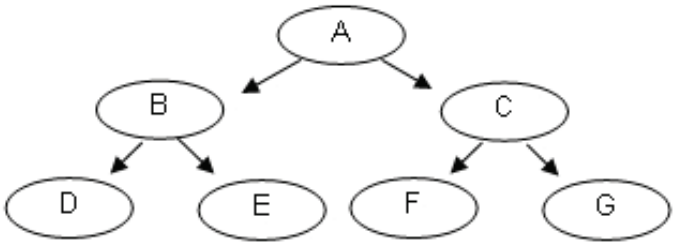


A → B → D → E → C → F → G

3.36. ábra Bináris fa preorder bejárása

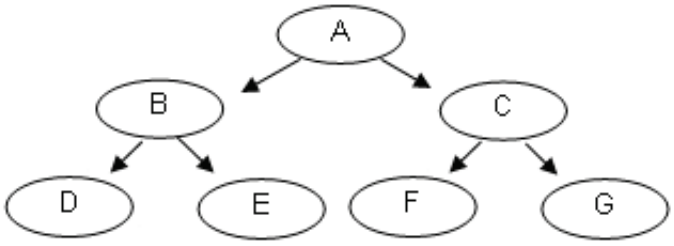
Inorder bejárás: ha a fa üres, akkor vége az eljárásnak. Ha nem, akkor járjuk be inorder módon a bal oldali részfát, dolgozzuk meg a gyökérelemet, végül járjuk be inorder módon a jobboldali részfát.

10 forrás: http://www.tankonyvtar.hu/hu/tartalom/tamop425/0005_22_algoritmizalas_alapjai_scorm_11/1135_fa.html



3.37. ábra Bináris fa inorder bejárása

Posztorder bejárás: ha a fa üres, akkor vége az eljárásnak. Ha nem, akkor posztorder módon járjuk be a bal oldali, majd a jobb oldali részfát, végül dolgozzuk fel a gyökérelemet.



3.38. ábra Bináris fa posztorder bejárása

A bináris keresőfa definíciója: A t bináris fát pontosan akkor mondjuk egyúttal bináris keresőfának, ha t bármely x csúcsára igaz az, hogy amennyiben y az x bal oldali részfájának egy csúcsa, illetve ha a z pont az x jobb oldali részfájának egy csúcsa, akkor

kulcs(y)<kulcs(x)<kulcs(z).

További érdekessége a bináris keresőfának, hogy inorder bejárás esetén rendezett sorozatot kapunk.

3.2.2. Tipikus keresési és rendezési algoritmusok

A legegyszerűbb adatstruktúra az egydimenziós vagy lineáris tömb, amelyben a tárolt elemek meg vannak számozva egymást követő egész számokkal, és ezekre a számokra hivatkozva érhető el a tartalom. A memóriában egymás után helyezkednek el.

Mint ahogy fent kifejtettük, az adatelemek tárolása nem egymást követően is kiosztásra kerülhet a memóriában, ilyenkor mutatókkal vannak összekapcsolva. Ezek a mutatók memóriacímeket tárolnak, melyek jelezik, hogy hol található a következő elem.

Sok algoritmust fejlesztettek ki az adatok hatékony válogatására, ezek specializálódhatnak a központi memóriában lévő struktúrákra, vagy akár az adatbázisokra is.

A rendezés többek között azért annyira fontos, mert sok más probléma egyszerűvé válik miután a lista elemeit rendeztük. A keresés felgyorsítása talán a legfontosabb alkalmazása a rendezésnek.

Az ismertebb rendezési algoritmusok:

- **Kiválasztásos rendezés**
- **Beszúrásos rendezés**
- **Buborékos rendezés**
- **Összefésüléses rendezés**
- **Gyors rendezés**

A keresés során **lineáris** vagy más néven **szekvenciális keresést, illetve bináris keresést valósíthatunk meg.**

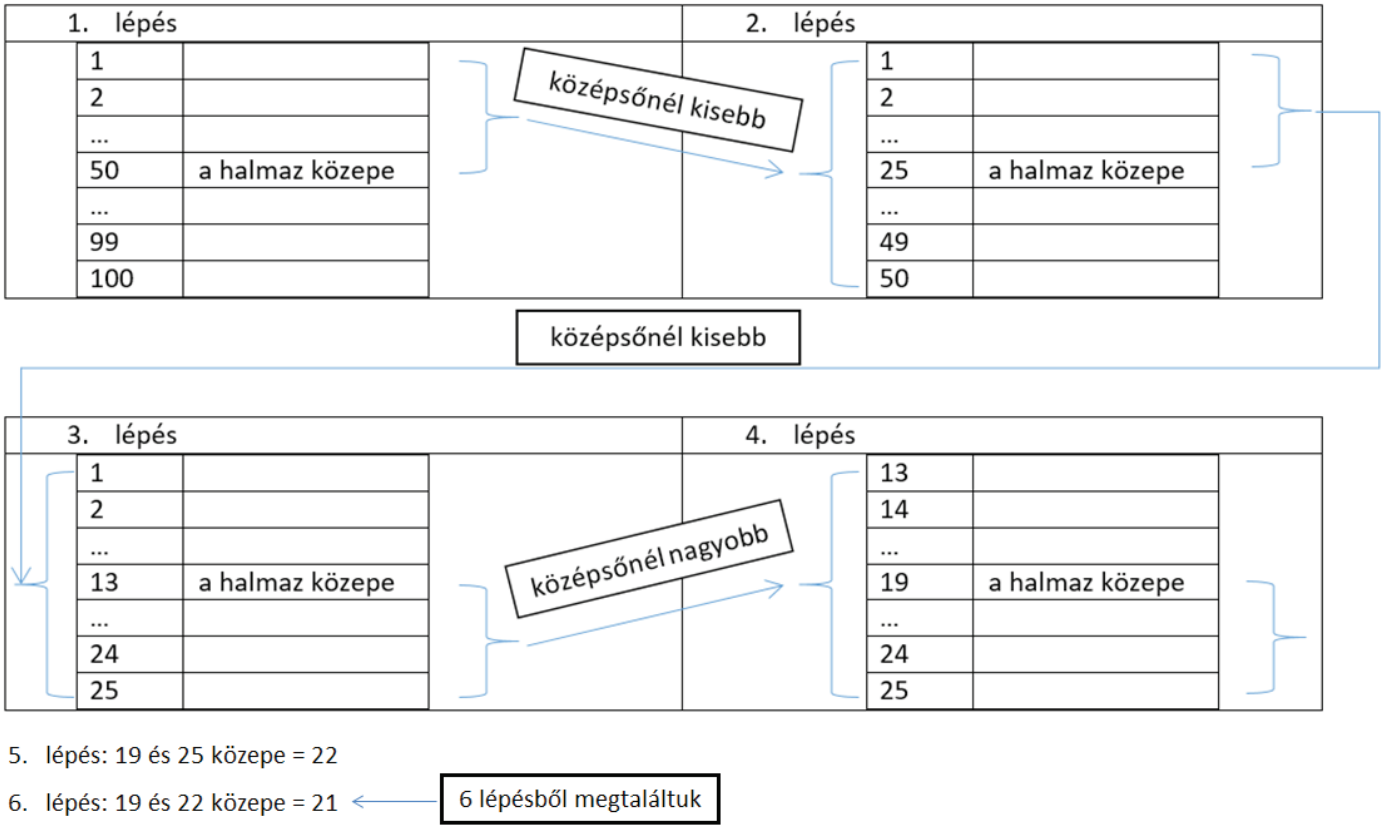
A lineáris keresés minden elemet megvizsgál, ezáltal jóval lassabb lesz. Amennyiben nincs rendezve a listánk, akkor csak lineáris keresést végezhetünk.

Ha a korábban felsorolt rendezések (vagy bármelyik másik) segítségével rendeztük a listánkat, használhatjuk a bináris keresést is.

A folyamat minden egyes lépésénél meg tudjuk felezni azt az adatmennyiséget, amiben keresünk.

Info **Bináris kereséssel ki lehet küszöbölni az adatok felét, minden egyes lépés során. A rendezett halmaz középső elemével hasonlítjuk össze a keresett elemet, vagy felette, vagy alatta lesz az elem (ha nem pont a középsőt kerestük), ezáltal a következő hasonlítást már csak az eredeti halmaz felén végezzük.**

Tegyük fel, hogy ki szeretnénk találni egy számot, 1-100 között. Ha nem ismerjük a bináris keresést, akkor össze-vissza tippelgetünk, vagy megyünk sorban. A legrosszabb esetben 100 lépés után kapjuk meg az eredményt. Keressük most a 21-et bináris módon:



5. lépés: 19 és 25 közepe = 22
6. lépés: 19 és 22 közepe = 21 ← 6 lépésből megtaláltuk

3.3. Programozási nyelvek

Tanulási célok

- Megkülönbözteti a programozási nyelvek főbb típusait, mint funkcionális, procedurális és objektumorientált, és bemutatja az előnyeiket.
- Bemutatja az eljárások és függvények alkalmazását, és különbséget tesz az érték és hivatkozás szerinti paraméterátadások között.
- Meghatározza a szintaxis fogalmát, és vázolja a jelentőségét a programozási nyelvekben.
- Különbséget tesz a programozási nyelveknél a fordítás (compilation) és az értelmezés (interpretation) között.

3.3.1. A programozási nyelvek főbb típusai

Funkcionális programozási nyelv

Ezt a programozási módot gyakran szembeállítják a procedurális programozással. A funkcionális programok általában kevésbé használják ki a tárolt állapotot, gyakran kerülik a ciklusokat, és inkább rekurzív függvényeket használnak.

A funkcionális programozás elsődleges fókusza a függvényértékek visszaadása, a mellékhatások és egyéb módok állapotának tárolása erősen ellenjavallt. Vannak kevert és vannak tisztán funkcionális nyelvek. A tisztán funkcionális nyelvben például elvárható, hogy ha meghívunk egy függvényt, akkor az ne módosítsa a globális változókat, vagy a kimeneteket. Annál is inkább, mert funkcionális megközelítésben mindent függvénynek tekintünk, még a konstansokat is. Változókat nem is használnak ezek a nyelvek.

Lehet azonban rekurzív hívásokat végrehajtani, és ezen hívások paramétereit megváltoztatni.

A funkcionális nyelvek általában egyszerűbb szintaktikával rendelkeznek, és könnyebb velük dolgozni elvont problémákon, de könnyebb „eltávolodni a géptől", mivel a programozási modell miatt nehéz megérteni, hogy pontosan hogyan van lefordítva a kód a gép nyelvére. Ez problémát okozhat rendszerprogramozás során.

Általában matematikai célú problémák megoldására használják, a mesterséges intelligencia kutatásában előszeretettel alkalmazzák. Az első funkcionális programnyelvet LISP-nek nevezték. A CAD-es programok AutoLISP nyelven vezérelhetőek. Eredetileg az AutoCAD program részére készült.

Procedurális programozási nyelv

A procedurális nyelvek állítások sorozatát hajtják végre, és ez vezet az eredményhez. Lényegében a procedurális nyelv a probléma megoldásához követendő eljárást fejezi ki. Tipikusan a strukturált programozási módszertan elemeit használja. A procedurális nyelvek általában sok változót használnak, intenzíven használnak ciklusokat és más állapotelemeket, ez különbözteti meg őket a funkcionális programozási nyelvektől. A procedurális nyelvekben a változók módosíthatók, vagy lehetnek más mellékhatásaik (pl. nyomtatási információ, globális változó értékének megváltoztatása).

Objektumorientált programozási nyelv

Az objektumorientált programozás olyan tárgyak gyűjteményének látja a világot, amelyeknek belső adataik vannak, és külső eszközökkel lehet hozzáférni ezekhez az adatrészekhez. Mondhatjuk, hogy a külső tárgyakból azok belső tulajdonságai alapján képezünk (absztrahálunk) objektumokat. Ezekhez a belső tulajdonságokhoz meghatározott hozzáférést engedélyezünk. A hozzáférés természetesen csak más objektumokra nézve kerül definiálásra. Az objektumorientált programozás célja, hogy egy problémára úgy tekintsünk, hogy felosztjuk őket objektumok gyűjteményére, melyek az adott probléma megoldására szolgálnak. Az objektumorientált programozásnak 3 alappillére van.

- Egységbezárás / adatrejtés
- Öröklés
- Polimorfizmus – többalakúság

Az egységbezárás lényege, hogy mindent egy osztályba (és ezáltal a belőle létrehozott objektumba) teszünk, ami jellemzi a modellezendő valós világbeli tárgyat, eszmét. Egy osztály adatait elrejthetjük a többi osztály elől, vagy korlátozhatjuk a hozzáférést. A műveletek tekintetében pedig csak a „mit csinál" kérdésre kapunk választ, a „hogyan csinálja" az osztály magánügye (a programozó által meghatározott mód) marad.

Az öröklés során a szülő tulajdonságait, műveleteit örökli az utód. Ezzel jelentősen csökkenthető a kód mérete, és ami ennél is fontosabb, a kód ismétlése is elkerülhető. Öröklés során lehetőségünk van kibővíteni az utód lehetőségeit mind új tulajdonságokkal, mind új metódusokkal. Arra is lehetőségünk van, hogy más működést definiáljunk az utód őstől örökölt metódusában. Ezt hívják felüldefiniálásnak.

A polimorfizmus során az ős tud úgy viselkedni, mint a belőle származtatott utódok. Ezáltal behelyettesíthetjük az őst oda, ahova egyébként a különböző utódokat kellene. Az ős(re hivatkozó mutató) polimorf módon viselkedik.

Szkriptek

A szkript nyelvek gyakran procedurális programozást jelentenek, melyek tartalmazhatnak objektumorientált nyelvi elemeket is, de külön kategóriába esnek, mert általában nem tekinthetők olyan teljes értékű programozási nyelvnek, melyek nagy rendszer fejlesztését támogatják. Például nem rendelkeznek fordítás közbeni ellenőrzéssel, és nem feltétlenül igénylik a változók típusának megadását. Pl. a 3D MAX program saját script nyelven vezérelhető.

3.3.2. Eljárások és függvények alkalmazása, paraméterátadási megoldások

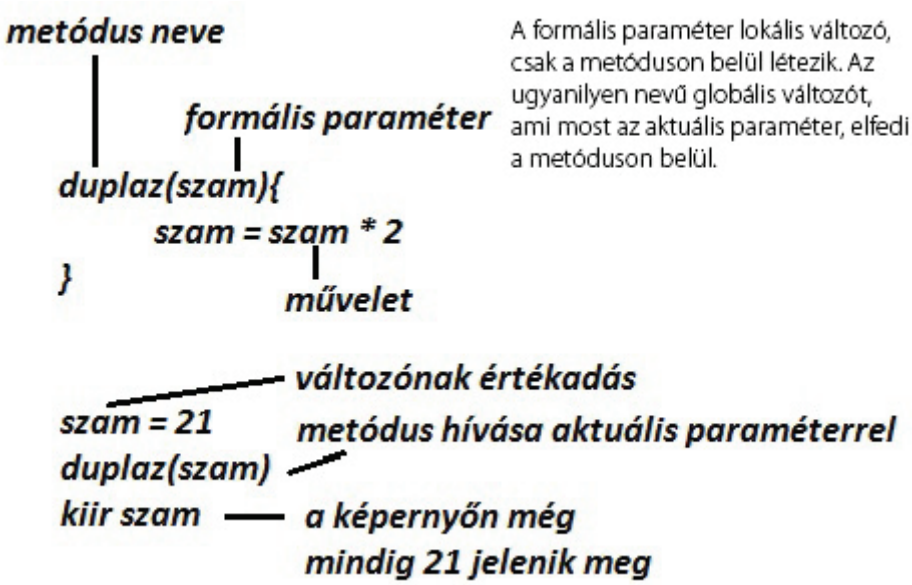
Amikor a teljes programot, vagy annak egy részét, például az adatbázis elemeinek grafikus megjelenítését másik programban is felhasználnánk, moduláris módszertant használunk. A felhasználandó részeket külön modulokba szervezzük. Gyakran előfordul az is, hogy a programunkon belül bizonyos műveleteket, például egy számítást többször kell elvégezni, más-más bemeneti értékekkel. Ezt is felfoghatjuk külön modulként, de ilyenkor azt mondjuk, hogy a műveletet kiszervezzük egy metódusba. A metódus lehet eljárás vagy függvény. Az alapvető különbség az, hogy az eljárás elvégzi a dolgát, és nem ad semmilyen visszajelzést, a függvény pedig a dolga végeztével úgynevezett visszatérési értéket küld.

A metódusok többnyire különböző bemeneti értékekkel végzik a műveletüket. Nem sok értelme lenne az „összead" nevű metódusnak, ha mindig csak a 2 és 3 számokat tudná – amúgy helyesen – összeadni. Ezért a metódusokat úgy készítjük, hogy minél univerzálisabbak legyenek. Az összeadás esetében ez azt jelenti, hogy kettő tetszőleges számot tud összeadni. A még tökéletesebb eredmény érdekében felkészíthetjük a metódusunkat változó számú bemenő érték feldolgozására is.

Ezek a bemenő adatok az argumentumok, vagy paraméterek. Természetesen lehet olyan metódusunk, ami nem kap paramétert. Amikor felhasználunk egy metódust, akkor azt mondjuk, hogy meghívjuk a függvényt vagy metódust. A hívás során aktuális paraméterekkel látjuk el a formális paramétereket. A matematikában a sin(x) függvény esetében a sin a függvény neve, az x a formális paraméter. Amikor meghívjuk, azaz használjuk, akkor a formális paramétert aktuális paraméterre cseréljük: sin(30), sin(p)

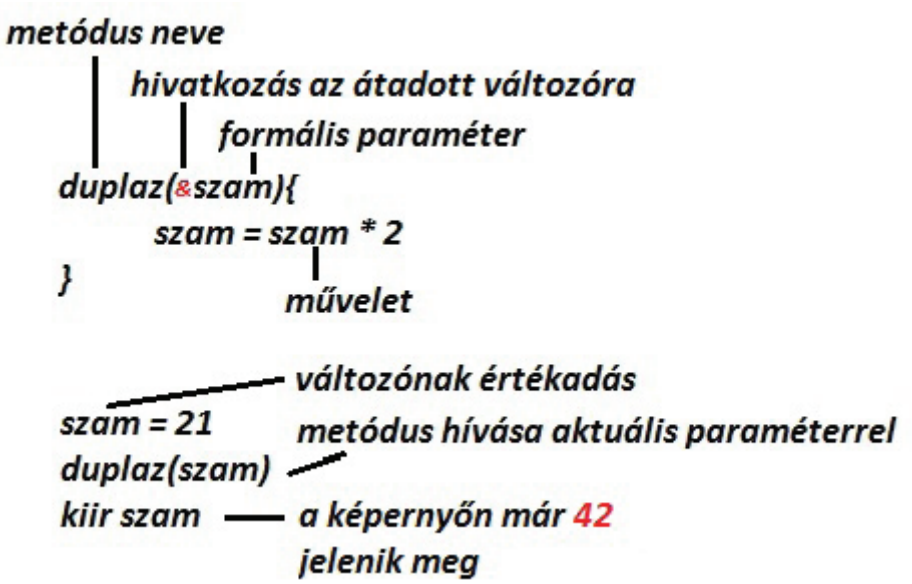
A sin(x)–et függvénynek hívjuk, mert kiszámolja és megmondja az értéket. Ebben az esetben az eljárásnak nem sok értelme lenne, mert kiszámolná, de nem tudatná velünk az eredményt. Eljárás lehet, ha az elvégzett műveletet pl. látjuk. Mondjuk arrébb tol valamit a képernyőn. Paraméterként megkaphatja a távolságot.

A paraméter átadásának kétféle módja létezik. Amikor érték szerint adunk át, akkor a metóduson belül úgynevezett lokális változók keletkeznek. Ez azt jelenti, hogy csak a metóduson belül létezik a (metódusra) lokális változó, csak itt lehet rájuk hivatkozni. A metóduson kívül nem érzékeljük a hatását. Az alábbi ábra magyarázza a működést:



3.39. ábra érték szerinti paraméterátadás

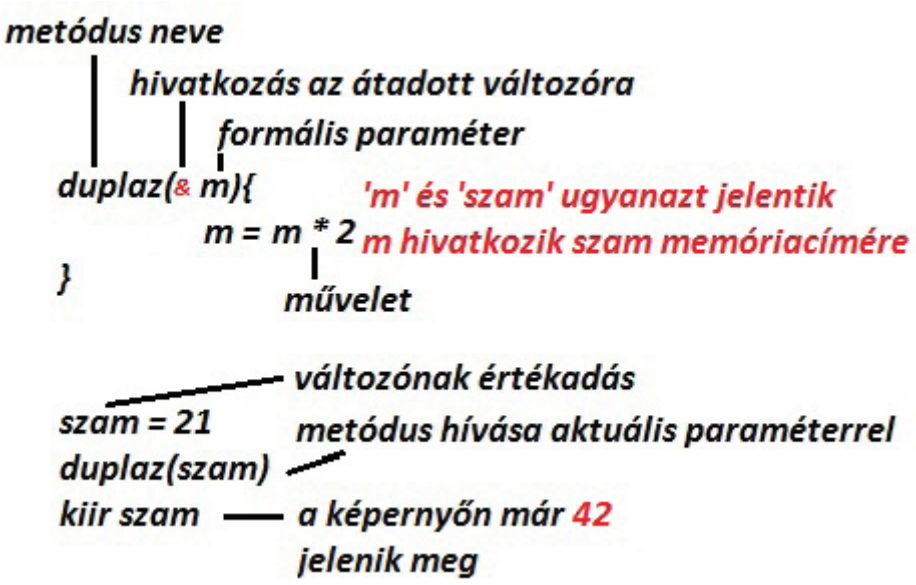
A másik módszer, amikor cím szerint adunk át paramétert. Ez azt jelenti, hogy nem készül másolat az átadott paraméterről a metóduson belül, hanem a paraméternek a memóriában elfoglalt címére készül hivatkozás. Bizonyos programnyelvekben azt mondjuk, hogy referenciát adunk a változóra. Az alábbi kódrészlet magyarázza a működést:



3.40. ábra cím szerinti paraméterátadás

A formális paraméterek elfedik a metóduson kívül deklarált változókat, ha ugyanaz a nevük. Ebből az következik, hogy formális paraméternek eltérő neveket is lehet használni. Hiszen érték szerinti átadás esetén (fenti ábra) lemásolódik az átadott érték a formális paraméter nevével megegyező nevű lokális változóba. Cím szerinti átadás esetén a formális paraméter nevével megegyező változó hivatkozni fog a metóduson kívül deklarált változóra. Tulajdonképpen ugyanarra a memória-területre hivatkozik, csak különböző a nevük.

Minden paraméter lehet konkrét érték, változó, konstans vagy referencia érték. Természetesen megfelelő módon kell definiálni a függvényt, de elvi akadályok nincsenek.



3.41. ábra cím szerinti paraméterátadás eltérő névvel

A moduláris programozást a függvények és eljárások használata teszi lehetővé. A függvényeket és eljárásokat úgy kell megírni, hogy különböző paraméterekkel minél több esetben működjenek. Ezeket a kódokat egy külön fájlba, modulba szervezhetjük (.dll). Ezáltal elkerüljük a kódismétlést.

3.3.3. A szintaxis fogalma és jelentősége

A szintaxis a programozási nyelv helyesírására és nyelvtanára utal. Érthetjük úgy is, hogy olyan szabályok, amelyek érvényes programokat hoznak létre. A számítógépek merev gépek, csak azt értik meg, amit pontosan abban a formában írunk meg, amire a számítógép számít. Ezt az elvárt formát nevezzük szintaxisnak. Ha egy program nincs helyesen megírva, akkor nem fog működni. A szintaxisnak helyesnek kell lennie.

Minden programozási nyelv meghatározza a saját szintaktikai szabályait, ami szabályozza, hogy milyen szavakat ért meg a számítógép, mely szavak kombinációjának van értelme, és milyen írásjelekre van szükség.

Példaként vegyünk egy parancsot, ami utasítja a programot, hogy rajzoljon egy *Kör*t aminek 10 a sugara. Ez azt feltételezi, hogy van egy *Rajz* és *Kör* nevű *Parancs*unk, ami meghatározza, milyen formájú alakzatot hozzon létre. A szintaxis szabályainak megfelelő parancs valahogy így nézhet ki: Rajz [Alakzat,][Méret]

Egy programnyelv megtanulása jelentheti azt, hogy megismerjük a nyelv szintaxisát. Programozni megtanulni azt jelenti, hogy az adott nyelv helyes szintaxisával szemantikailag is helyes programot írunk. A szintaxis hibáira figyelmeztetést kapunk, mert nem tudja értelmezni a program. Azonban a szemantikai hibák a gép számára nem hibák, hiszen tudja értelmezni a parancsot, csak nem azt csinálja, amit mi szeretnénk. Egy képlet beírásánál, ha lehagyjuk a zárójelpárokat, attól az a képlet még számolható a gép számára, de nagy valószínűséggel hibás eredményt fog adni. Ha csak a zárójel-pár egyik felét hagyjuk le, az szintaktikai hiba, hiszen nem lehet értelmezni, hogy hol szerettük volna bezárni. Csak azt tudja a fordító vagy értelmező, hogy a zárójeleknek párban kell lenniük.

3.3.4. A fordítás (compiler) és értelmezés (interpreter) közötti különbség

Ahhoz, hogy egy program fusson a számítógépen, a programot először le kell fordítani a gép számára is érthető nyelvre. Ehhez ismerni kell a processzor utasításkészletét. A programozó által készített programleírást forrásprogramnak, a gép (processzor) által értelmezhető kódot gépi kódnak nevezzük. A forrást le kell fordítani, ekkor úgynevezett tárgykód keletkezik, amiben lehet gépi kód, de nem tudja közvetlenül értelmezni a processzor. Különböző nyelveknél különböző eljárások vannak, de a tárgykód tovább alakul gépi kóddá. A különböző nyelvek fordító vagy értelmező használatával teszik használhatóvá a forráskódot.

Fordítás (Compiler)

Ha megtörtént a szintaktikai ellenőrzés, akkor létrejön a forráskód lefordított változata, a tárgykód. Ha nemcsak egy tárgykóddal dolgozunk, a következő lépés a tárgykódok összefűzése, C, C++ nyelveknél ezt a szerkesztő (linker) végzi. Ebből lesz a gépi kód.

C# nyelven egy úgynevezett köztes kód, az IL (néha CIL–nek mondják) kód keletkezik. A .NET keretrendszer értelmezi és futtatja ezt a kódot.

Java nyelvnél a forrás a .java fájlokban található, ebből készül a .class fájl. Ez hívhatjuk tárgykódnak. Ezt a kódot a JVM – Java Virtuális Gép – értelmezi és futtatja.

Értelmezés (interpreter)

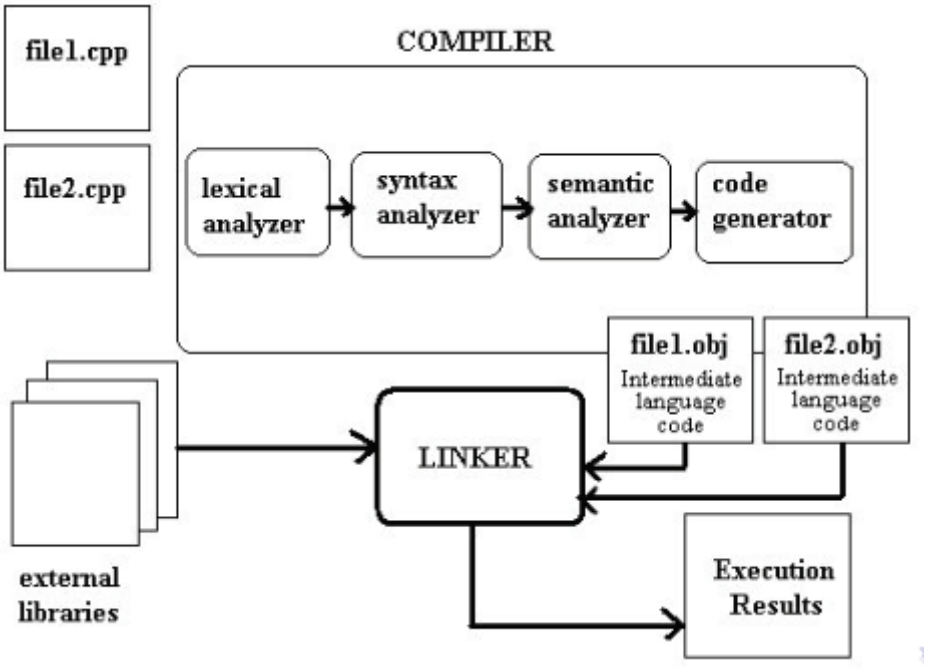
Az értelmezés során a forrás futási időben kerül értelmezésre, azaz, ha szintaktikailag rendben volt, akkor az értelmező átalakítja gépi kóddá. Ez a művelet sorról sorra hajtódik végre.

Minden egyes programozási elvnek vannak előnyei és hátrányai.

Az interpretált nyelveken többnyire könnyebb fejleszteni, látványosabb az eredmény. A visszajelzések is emberközelibbek. Ugyanakkor lassabbak, hiszen minden futtatáskor az értelmezőnek ellenőriznie kell a szintaxist és át kell alakítania gépi kóddá. Szinte beszélt nyelven lehet bennük programot írni.

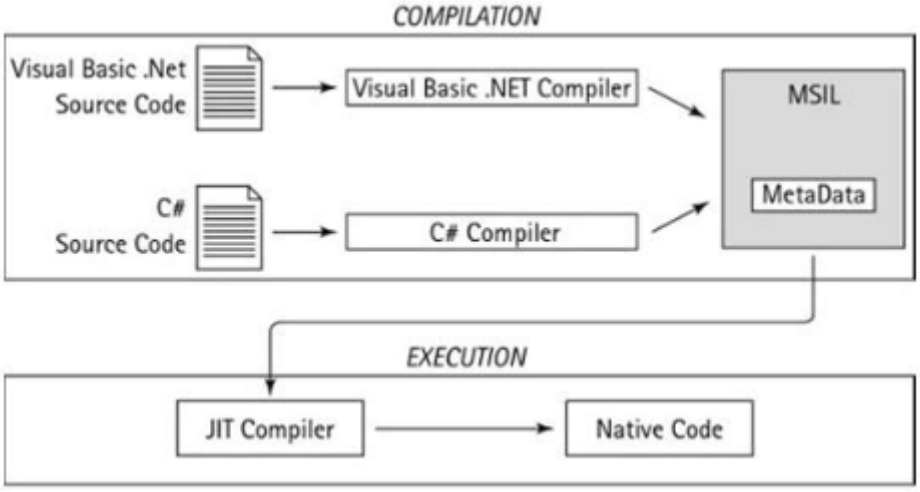
Ezzel szemben a fordított programok gyorsak, mert a fordításkor a processzor számára értelmezhető kód kerül eltárolásra. Így nem hordozható a program, másik gépnél általában újra kell fordítani. A Java és C# nyelvek a köztes kódjukkal orvosolják ezt a problémát. Maga a fordítás időigényes, a fejlesztés komoly programozói ismereteket igényel.

Összehasonlításként[11]

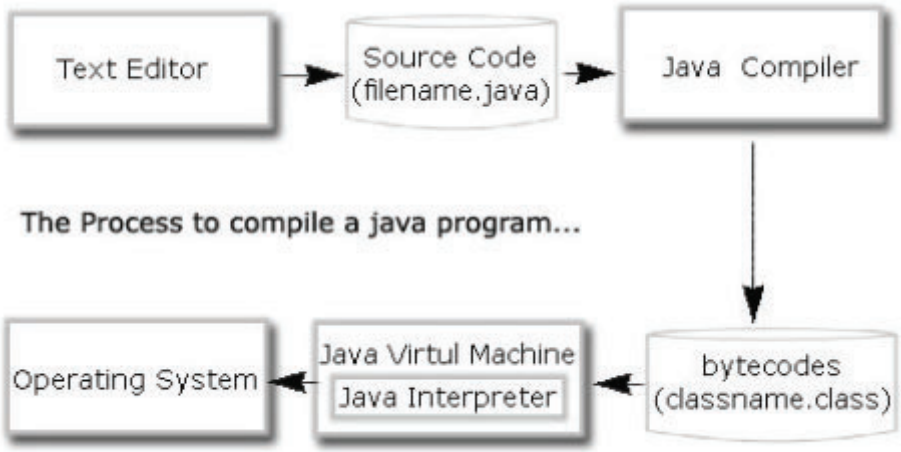


3.42. ábra Hagyományos fordítás

11 képek forrása: http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/krizsanz/Tantargyak/oop/bin/01_dotnetAlapok.pdf



3.43. ábra Just in Time (JIT) fordítás



3.44. ábra Java interpretálás

Natív fordítás (pl. C++)	JIT fordítás (MS) (pl. C#)	Interpretált futtatás (pl. Java)
Nagyon gyors. Mindent meg kell írni. Minden platformra le kell fordítani. Memóriát kell kezelni. Futásához nem kell semmi.	Beépített mechanizmusok. Közepesen gyors. Nem platform független. Windows rendszeren fut. Több támogatott nyelv. Sok kész osztály. Keretrendszer kell a futásához.	Nagyon lassú. Teljes platform függetlenség. Sok kész osztály Meg kell tanulni a Java-t. Sok kész osztály. Virtuális gép kell a futásához.

3.45. ábra Interpretáélás vs. JIT natív futás

3.4. Objektumorientált programozás

Tanulási célok

- Bemutatja az objektumorientált tervezés alapjait.
- Bemutatja az objektumorientált programozás elvét.
- Bemutatja az osztály, az objektum, a példány és a metódus fogalmakat és ezek egymással való kapcsolatát az objektumorientált programozásban.
- Bemutatja az öröklődés fogalmát, és azt, hogy az milyen lehetőséget kínál a programozónak.
- Bemutatja az absztrakció és az egységbezárás (információelrejtés) elvét.
- Bemutatja, hogy a polimorfizmus (többalakúság) az újrahasználható komponensek fejlesztése révén hogyan segíti elő a hatékony programtervezést.

Az objektumorientált tervezés alapjai

Az objektumorientált módszerek a szoftverfejlesztés életciklusának különböző fázisaiban, az elemzés, a tervezés és a kivitelezés során egyaránt alkalmazhatók.

Az objektumorientált tervezés (OOD, azaz Object-Oriented Design) az a folyamat, amelyben a fejlesztők elvégzik azt az absztrakciót, amely az elemzés során meghatározott követelményeknek megfelelően leképezi a rendszer viselkedését, és kidolgozzák a viselkedés modellezéséhez szükséges műveleteket.

Az OOD viszonylag független a programozó által használt programozási nyelvtől. A módszer nem más, mint a szoftver architektúra lebontása olyan objektumokra, amelyekkel a rendszer műveleteket végez.

A**z objektumorientált tervezés alapvető fogalmai:**

- Lebontás/Felépítés
- Absztrakció
- Szétválasztási szabályok és műveletek
- Részhalmazok és kód-családok azonosítása
- Újrahasznosítás

A tervezési módszer fő célja a szoftverbonyolultság kezelése a szoftverminőséget meghatározó tényezők javítása révén.

Az OOD a rendszer statikus összetevőit írja le. Ez az osztály- és objektum-diagramok megadásával lehetséges.

Az OOD célkitűzései:

- A rendszer modulokra bontása és az architektúra meghatározása csoportosítással.
- A modulok között kapcsolatok és függőségek azonosítása.
- A modul-interfészek specifikálása, a modulok egymástól független tesztelése és a csoportos kommunikáció javítása érdekében.
- A modulok funkcióinak formális (a modulinterfész-specifikációkkal), és informális leírása (kommentekkel és dokumentálással).

Az alábbi modellek szempontjából vizsgálhatjuk a teljes rendszert:

A statikus modell a nem változó komponensekről szól, a „kik a résztvevők" kérdésre ad választ.

A dinamikus modell az időbeliség kérdését járja körül. A rendszer dinamikáját írja le, azt, hogy a statikus modellben szereplő elemek mikor végzik a dolgukat.

A funkcionális modell lesz az, ami a rendszer funkcióit, tehát a „mit csinál" kérdésre választ ad.

- Statikus – ki?
- Dinamikus – mikor?
- Funkcionális – mit?

Miután megvannak a modelljeink, elő kell állítanunk egy olyan dokumentációt, ami konkrétan realizálja a megvalósítás lehetőségeit, folyamatát. Ezt hívjuk tervezésnek. A tervezés a konkrét programozás előtti lépés, bele lehet venni az adott nyelv sajátosságait. Továbbá figyelembe kell venni az architektúrát és a külső rendszerekkel való kommunikációt is.

3.4.1. Az objektumorientált programozás elve

Az objektumorientált programozás (OOP - Object-Oriented Programming) alapfogalmai közé tartozik az absztrakt adattípus, az öröklődés és a dinamikus kötés. A program hierarchiába szervezett objektumok közötti üzenetekből áll. Az OOP nyelvek elődei a szimulációs programnyelvek. A OOP fogalmi modell alapgondolata az, hogy a rendszer képes reagálni az általa „szimulált" környezet változásaira. Ha például egy ipari folyamatot modellezünk OOP-ben, a folyamat minden fontos eleméhez hozzá kell rendelnünk egy objektumot. Az objektumok üzenetekkel lépnek kapcsolatba egymással. Az OOP filozófia középpontjában nem a programkód, nem az algoritmusok, hanem az adatok állnak. Az objektumorientált programozás a valós világ dolgainak kölcsönhatását az objektumok közötti üzenetváltással modellezi.

Míg az OOP elsősorban programnyelvekkel és az implementáció kérdéseivel, az OOD az alkalmazás tervezésével foglalkozik.

OOP alapfogalmak

Az adatabsztrakció és információ elrejtés elve

Az absztrakció az a folyamat, amikor egy jelenség vagy objektum lényegtelen tulajdonságaitól elvonatkoztatunk. Az adatabsztrakció során be tudjuk határolni az összetett adatobjektum használati eseteit, átvizsgálva, hogy az hogyan épül fel más, egyszerűbb adatobjektumokból.

Az adatabsztrakció alapgondolata az, hogy a programot az összetett objektumok használata szerint strukturáljuk, amely így absztrakt adatokkal „tud" dolgozni. Nézzük ezt egy példán keresztül: a munkáltató munkavállalóként tekint az emberekre. Számára a munkavállaló életkora, a neme, a munkatapasztalata, a képzettsége, egészségügyi állapota fontos. Nem foglalkozik a dolgozó ruhájának színével, a kapcsolataival, vagy azzal, hogy milyen ételt eszik. Az absztrakció az objektumorientált szoftverfejlesztés kiindulópontja: meghatározza a rendszer szempontjából legfontosabb szempontokat. A valóság leegyszerűsítésével a lényeg kiemelésére törekvő szemléletmód az absztrakció. Egy adott rendszer elemzése során végzett absztrakciót modellezésnek nevezzük.

Más esetben a dolgok **specializálás**ára van szükség. Az autógyárban a végső összeszereléskor nem mindegy, hogy a vevő által megrendelt gépkocsi mekkora motorral, milyen színre festve, milyen kárpittal és milyen felszereltséggel készül el, tehát itt már fontosak a részletek.

A modellezés minden esetben absztrakciós (általánosítás) és specializáló (részletező) feldolgozásból áll. Az elkészült modellek egymáshoz nagyon hasonlíthatnak. Például: minden személyautó négykerekű. Az ugyanolyan tulajdonságokkal jellemezhető, vagy ugyanúgy viselkedő modelleket egy csoportba, vagy **osztályba** lehet sorolni. Az osztályozás jelentősége abban rejlik, hogy elegendő az azonos jellemzőket egy helyen, az osztály definiálásakor (osztályszinten) megadni. Az osztály tehát egy a valóságos tárgy vagy fogalom modellje, ami a számunkra fontos jellemzőit és képességeit írja le. Az objektum az osztály modellje alapján készült példány, ami a memóriában tárolódik és képes az objektum jellemzőit is tárolni.

Az objektumban tárolt adatok így már a valódi tárgy jellemzői. Példaként: az *Autó* osztály egy tulajdonsága, hogy van valamilyen színe. A *Szín* tulajdonság konkrét értéke az osztály egy példányában, az objektumban van tárolva (például az, hogy a sárga).

Az információ elrejtés elve

Azt jelenti, hogy egy szoftverkomponens használójának csak annyit kell tudnia egy objektumról, hogy hogyan kell azt létrehozni, kezdőértékeit beállítani (inicializálni), és hogyan lehet az adatait (tulajdonságait) elérni. Az objektum programkódjához a programozó nem férhet hozzá. Az elrejtés elve szerint az

objektum adatait csak az objektum saját eljárásai kezelhetik. Az objektumok adatai így az objektum „tudta" nélkül nem változtathatók meg.

Aktív (és nem passzív!) típusok

Az OOP az adatokat, amelyekkel a függvények és eljárások dolgoznak, nem passzív, hanem olyan aktív elemeknek tekinti, amelyek képesek a környezetükkel kapcsolatba lépni. Az imperatív programozás kontextusában a hangsúly a folyamatvezérlés leírásáról átkerül az objektumok közötti interakciók leírására.

Generikus programozás

Amikor egy objektum valamely függvényét meghívjuk, a hívót általában ügyfélnek (kliensnek), a hívást ténylegesen végrehajtó objektumot pedig kiszolgálónak szokták nevezni, mivel a hívó, mint kliens tulajdonképpen igénybe veszi az objektum valamely szolgáltatását.

A generikus programozásban vannak úgynevezett paraméterezett típusok, azaz az osztályokat és/vagy függvényeket nemcsak változókkal, hanem változó típussal is paraméterezhetjük. Ez lehetővé teszi, hogy a deklaráció során paraméterként adjuk meg annak az objektumnak a típusát, amivel az eljárásnak dolgoznia kell, és a kiértékelésre fordítási időben kerüljön sor. Gondoljuk el, ha szeretnénk rendezni egy számsort, akkor azt megírjuk egész (int) típusok kezelésére. Ha valós (float vagy double) típusokat adunk át, új metódusra van szükség. Itt még segíthet, hogy a valós kezeli az egészeket, de mondjuk az előbbi Cím típusunk utcanevei alapján történő rendezés már mindenképp új metódus írását követelné.

Szükségünk van egy úgynevezett sablonra, amit bármilyen típussal hívhatunk, ha az rendelkezik a sablonban hívott publikus metódussal:

```
template <typename T>
void metodus(T Objektum){
Objektum.PublikusMetodusa();
```

A fenti kódrészletben Objektum bármilyen T típusú lehet, csak legyen egy publikus elérésű PublikusMetodusa nevű metódusa.

Öröklődés és dinamikus kötés

Az objektumorientált programozásban az osztályok örökölhetik más osztályok állapotát és viselkedését. Képzeljünk el egy bicikli osztályt, amely a biciklik összes alaptulajdonságát leírja. Ennek az osztálynak lehetnek olyan alosztályai, mint a Mountain Bike, Trial Bike és a Tandem. Minden alosztály örökli a biciklik alaptulajdonságait, de mindegyiknek lehet valamilyen csak rá jellemző speciális tulajdonsága. Például a Tandem osztály esetén a két ülés. Az öröklést tehát egy már definiált osztály kibővítésének tekinthetjük.

Amikor egy metódust meghívunk, lehet, hogy az az ősben is megjelent, és az utódban felüldefiniáltuk. Ilyenkor fordításkor nem eldönthető, hogy melyik metódust kell végrehajtani, ez a döntés dinamikusan, futási időben fog megszületni. Késői kötésnek vagy dinamikus kötésnek nevezzük ezt a folyamatot.

Programozás szerződéssel

A szerződések (contractok) segítségével megkövetelhetjük, hogy bizonyos elő-, utófeltételek és invariáns tulajdonságok teljesüljenek egy metódus végrehajtásakor. A contractok előnyös tulajdonsága, hogy az adott metódus dokumentációjának részévé válnak, viszont a betartásuk ténylegesen kötelező, hiszen ezek a feltételek a működő kód részei, nem csak dokumentációs megjegyzések, tehát a kódrészletek a program futtatásakor végrehajtódnak, a feltételek ellenőriződnek. A futás idejű hibákat már igen korán, a kiváltódás helyének közelében diagnosztizálhatjuk contractok segítségével, és mindezt egy jól áttekinthető, egységes formában érhetjük el. Contractok írásakor és tesztelésekor a programozó jobban rá van kényszerülve, hogy átgondolja az általa írt osztály viselkedését, metódusait. A contractok emellett nem rónak túl nagy terhet a programozóra, hiszen öröklődéskor a contractbeli feltételek is öröklődnek, a megfelelő szabályok mellett.

A contract a következő kérdésekre ad választ:

A) mi a megfelelő pillanat a metódus meghívására?
B) mit várunk a metódustól, mit kell tennie?
C) meg fogja változtatni a metódus az objektum állapotát?

Kivétel-kezelés és érvényesítés (Exception handling and Assertions)

A kivétel a program végrehajtása során bekövetkező abnormális esemény, amely megszakítja a program normális futását. A programozók feladata meghatározni, hogy mi történjen, amikor egy kivétel bekövetkezik. Ha a kivétel nincs megfelelően kezelve, a program futása rendellenesen szakad meg, aminek számos negatív következménye lehet. Például nyitva maradhat a hálózati kapcsolat, az adatbázis-kapcsolat, nem záródnak le állományok, és emiatt az adatbázis, a fájl rekordjai inkonzisztens állapotban kerülhetnek. A kivétel-kezelés a programnyelvekbe beépített mechanizmus az ilyen hibák kiküszöbölésére.

Kivétel kezelése:

```
try{
        //ezt az utasítást próbáld végrehajtani
}
catch(…){
        //ha nem sikerült, ezt csináld
}
```

A try blokkba írt utasítás(ok) futás idejű végrehajtása során, ha nem sikerült a művelet, a vezérlés a catch blokkba kerül. Itt kezelhetjük a kivételt. Több catch ág is írható, a különböző kivételek kezelésére. Ilyenkor más típus kerül a catch paraméterébe

Az érvényesítés (assertion) a program logikájának tesztelésére szolgál. Beillesztünk a programba egy logikai kifejezést, amelyről azt feltételezzük, hogy értéke akkor lesz „IGAZ", amikor a program hibátlanul fut. Ha a kifejezés „HAMIS", a programfutás hibát jelez, figyelmeztet arra, hogy van egy hibás feltétel, amit meg kell keresni. Ez sokkal jobb megoldás, mint az „if-then-else" utasítás, mivel lehetővé teszi, hogy a programozó megfelelő módon dokumentálja a feltevéseit, és nem rontja a teljesítményt.

Érvényesítés:

```
assert(valtozo >= 0);
cout <<„Rendben lefutott" << endl;
```

Amennyiben „valtozo" értéke negatív, a program futása megáll. Csak pozitív érték esetén íródik ki a „Rendben lefutott" sor.

Az OOP legfőbb előnye a kód újrahasználhatósága. Az újrahasználhatóság azt jelenti, hogy egy meglévő programkódot a programozó újra felhasználja, növelve ezzel a hatékonyságot és egyszerűsítve a karbantartást. Ezzel a módszerrel egyfelől csökkentjük a fejlesztési és tesztelési időt, másrészt javítjuk a minőséget, hiszen egy másik alkalmazásban már kipróbált modult használunk.

3.4.2. Az osztály, az objektum, a példány és a metódus fogalma, valamint kapcsolataik

Az objektumorientált program legkisebb egysége az **objektum**, az objektumban az adatok és eljárások egységet alkotnak úgy, mint a valóságban (a kutya eszik, ugat, szalad, van színe, fajtája, súlya, stb.).

Az objektum információt tárol, kérésre feladatokat hajt végre. Az objektumokat – az objektumorientált program alapvető futás-idejű komponensei – adatok (tulajdonságok) és metódusok (operációk, műveletek) összessége alkotja. Az objektumok felelősek feladataik elvégzéséért.

A programozás feladata a probléma elemzése, az objektumok és köztük zajló kommunikáció természetének vizsgálata. A program végrehajtása közben az objektumok üzeneteken keresztül kölcsönhatásba lépnek egymással, miközben egymásról, egymás adatairól és programkódjáról semmilyen belső információval nem rendelkeznek. Minden objektum leírható tulajdonságokkal (az objektumra vonatkozó tulaj-

donságokkal) és műveletekkel. A tulajdonságokat az objektum attribútumainak, vagy adattagoknak, a műveleteket **metódus**oknak nevezzük.

Az **osztály** a hasonló típusú objektumok gyűjteménye. Miután egy osztályt meghatároztunk, tetszőleges számú, az osztályba tartozó objektumot hozhatunk létre. Az osztály meghatározza az osztályba tartozó objektumok tulajdonságait, és azokat a metódusokat, amelyeket az osztályba tartozó objektumok el tudnak végezni.

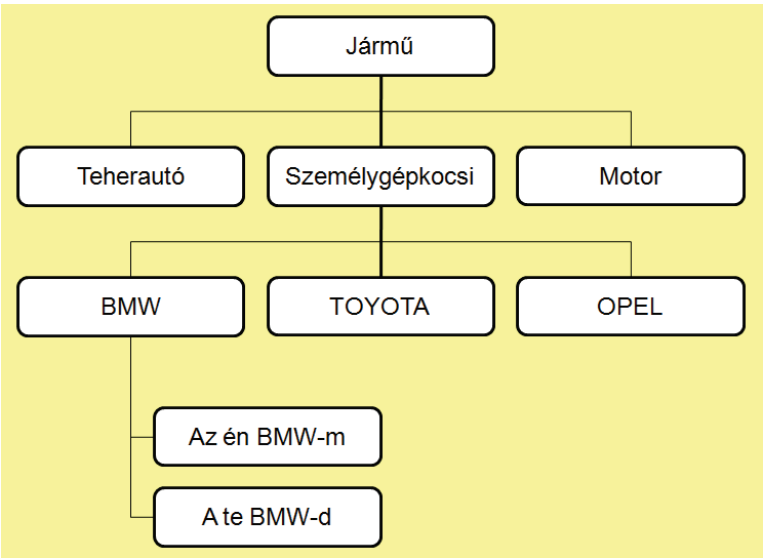
Az osztály absztrakció, az objektum pedig az osztály egy konkrét **példány**a. Egy adott objektum tulajdonságainak konkrét értéke határozza meg az objektum állapotát. Az objektum létrehozását – egy objektumváltozó deklarálását – példányosításnak is szokás nevezni.

Néhány alapszabály:

- Az azonos típusú objektumok osztályt alkotnak. Az osztály egy „sablon" amelyből objektumpéldányok hozhatóak létre. (Nem összekeverendő a sablon osztály fogalmával)
- Az objektumok azonosíthatóak. Két különböző objektum akkor sem azonos, ha tulajdonságaik egy adott pillanatban azonosak.
- Az objektum adatai a tulajdonságok. Ezek tervezési és/vagy futási időben kapnak értéket.
- Az objektumnak mindig van pillanatnyi állapota, amit a tulajdonságok konkrét értéke határoz meg. A metódusok végrehajtása után az objektum állapota megváltozhat.
- Az objektum állapotára csak azok a metódusok vannak hatással, amelyeket az osztályában leírtak.
- Az objektum adatait csak metódusokon (interfészen) keresztül lehet elérni, és az interfész a lehető legkisebb legyen.

3.4.3. Öröklődés fogalma, lehetőségei

Az öröklődés az objektumorientált programnyelv alapvető fogalma, célja a redundancia csökkentése. Bármely objektum osztályból utódokat képezhetünk. Az utód örökli az ős összes adatát, de bővülhet új tulajdonságokkal és metódusokkal is, illetve az örökölt metódusokat az utódban lehet módosítani. Összetettebb programokban az osztályok hierarchikus rendszert alkotnak. Az öröklődés (inheritance) segítségével egy-egy osztályból **alosztály**okat képezhetünk, az így létrejövő új osztály örökli az ősosztálya (használatos még a szülőosztály vagy szuperosztály elnevezés is) összes tulajdonságát. A leszármazott osztály elemei emiatt reagálnak az ősben definiált metódushívásokra. Az öröklődés során a leszármazott osztályok kiterjeszthetőek, új tulajdonságokkal (új változók, további eljárások) rendelkezhetnek.



3.46. ábra Öröklődés

A módszer a programozó számára idő és energia megtakarítást eredményez, hiszen nem kell újra meg újra ugyanazt a programrészletet megírni, az utód osztály kódolásakor felhasználhatja az ős osztályhoz megírt kódot. Az öröklődés automatikusan létrejön, amikor a programozó meghatározza, hogy egy osztály mely

másik osztály alosztálya. Az öröklődés meghatározása révén alakul ki az osztályok közötti hierarchia. A hierarchiában alacsonyabb szinten lévő osztályok öröklik a magasabb szintű osztályok tulajdonságait és metódusait. Az öröklődés az objektumok közötti kapcsolatok egyik fajtája.

3.4.4. Az absztrakció és az egységbezárás (információelrejtés) elve

A minket körülvevő világ végtelenül összetett és bonyolult. A modellezéskor nem feltétlenül kell teljes részletezettséggel megadnunk az objektumok minden paraméterét (valószínűleg nem is tudnánk). A legtöbb esetben elegendő egy nagymértékben egyszerűsített modellt készítenünk. A valóság leegyszerűsítésével a lényeg kiemelésére törekvő szemléletmód az **absztrakció**.

Az absztrakció legáltalánosabb esetei:

- Névabsztrakció
- Kifejezés-absztrakció
- Eljárás-absztrakció
- Adatabsztrakció
- Vezérlés-absztrakció

Az egységbezárás

Az egységbezárás (encapsulation) és az adatrejtés (data hiding), igen gyakran összemosódó objektum-orientált alapfogalmak. A bezárás, amire már korábban utaltunk is azt mondja ki, hogy az adatokat és a rajtuk végezhető műveletekért felelős eljárásokat egy egységként kell kezelni. Minden eljárást úgy kell meghatározni, hogy az csak az objektum saját adataival végezhessen műveleteket. Ehhez szorosan kapcsolódik az adatrejtés, ami pedig azt mondja, hogy az objektumoknak el kell rejteniük a külvilág felől az adataikat,az adatokhoz mindig csak valamilyen ellenőrzött formában lehessen hozzáférni.

Az objektum-orientált tervezési ajánlás az, hogy az objektum adatait csak az objektum saját eljárásai (metódusai) kezelhetik. Az objektumok adatai így az objektum „tudta" nélkül nem változtathatók meg. Ennek előnyei akkor mutatkoznak, ha egy objektum adatát csak úgy lehet megadni, hogy előtte az adat érvényességét és a többi adattal való összefüggését megvizsgáltuk az objektum saját metódusában. Autós példánál maradva, ne lehessen az autó (objektum) színét beállítani lilára, mivel ilyen színt nem akarnak gyártani. Ha mégis a programban ez előfordulna, akkor az objektum képes a hibás adattárolási kérelmet feldolgozni és esetleg hibaüzenetet küldeni.

A két fogalom jelentőségét együttesen lehet meghatározni. Ha az objektum zárt, és a külvilág felé rejtett, akkor más programrész nem tudja elrontani az objektum belsejét, ill. fordítva is igaz, az objektum belsejében lévő hiba nem befolyásolja a többi programrészt.

Az egységbezárás módszere kettős előnnyel jár. Egyfelől lehetővé teszi a program moduláris felépítését, másfelől pedig kihasználja az információ elrejtéséből származó előnyöket. A moduláris felépítés kiváló tervezési megközelítés, hiszen lehetővé teszi, hogy a fejlesztő kezelni tudja a rendszer bonyolultságát a „decentralizált szoftver architektúra" alkalmazásával.

A fejlesztés skálázhatóvá válik, a különböző programrészeken különböző programozók dolgozhatnak egymással párhuzamosan. A moduloknak jól definiált, absztrakt, de szűk interfésszel és magas belső kohézióval kell rendelkezniük. A modul összes belső információjának védettnek (private) kell lennie, kivéve azoknak, amelyeket publikusnak (public) deklarálunk.

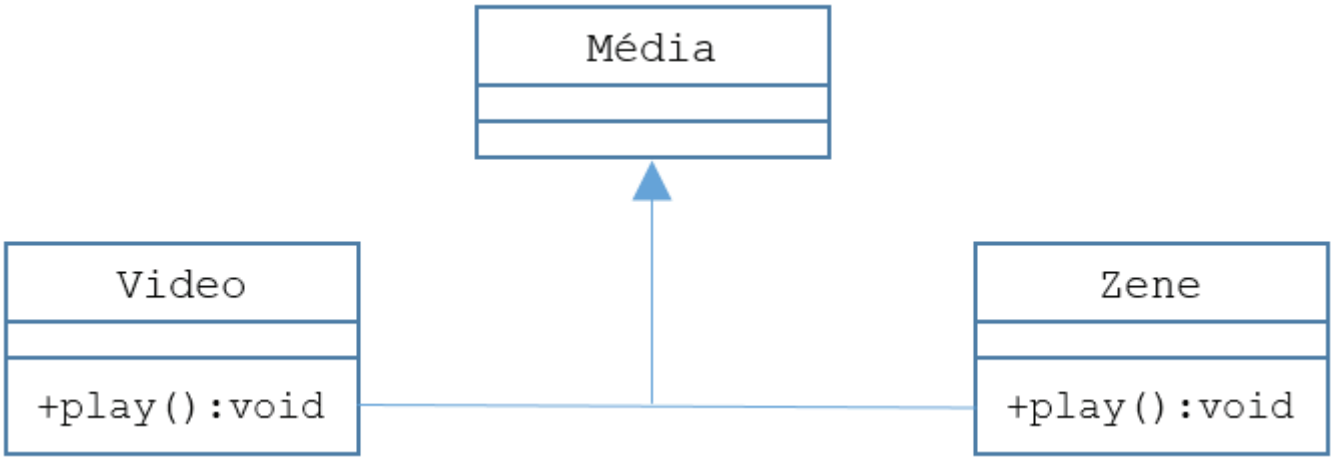
Amit el kell rejteni:

- az adatok reprezentációi
- algoritmusok
- input/output formátumok
- szabályok és műveletek
- alacsonyabb szintű modul-interfészek

3.4.5. A polimorfizmus (többalakúság) használatának előnyei

A polimorfizmus összefügg az öröklés fogalmával. A polimorfizmus lehetővé teszi számunkra, hogy különböző metódusokra ugyanazzal a névvel hivatkozzunk, annak ellenére, hogy azok eltérő műveleteket végeznek az objektumokkal.

Ez azt jelenti, hogy ugyanarra az üzenetre más-más objektumok másként reagálnak. Legyen például az üzenet az, hogy „játszd le" (PLAY). Mást jelent a lejátszás, ha az objektum egy zeneszám, és megint mást, ha egy videóklip.



3.47. ábra Polimorfizmus

A Média ősosztályból öröklődik a Video és a Zene osztály. Mást fog csinálni mindkettő play metódusa. Az ábrán a + jel azt jelenti, hogy publikus a metódus, a void jelentése, hogy nincs visszatérési értéke a függvényeknek. A teli fejű nyíl jelenti az öröklést az UML ábrákban.

A polimorfizmus tervezési elv azt jelenti, hogy azonos üzenetre különböző objektumok eltérő „viselkedéssel" reagálnak. A viselkedés attól függ, hogy milyen típusú az objektum, amely az üzenetet fogadja. A polimorfizmusra is tekinthetnénk úgy, mint interfészre, de külön fogalmakról van szó. Amíg az interfész lehetőségek gyűjteménye, amit majd a konkrét megvalósító osztály kifejt, azaz definiál, hogy a lehetőséget pontosan hogyan valósítja meg, addig a polimorfizmus inkább másfajta viselkedést jelent. Az objektumra hivatkozó mutató (akkor is, ha a nyelv nem használ mutató típust) más típusú (utód helyett ős) objektumra hivatkozik. Az előzőek figyelembevételével mondhatjuk, a polimorfizmus: egy interfész, de többféle módszer.

A polimorfizmus következménye az, hogy a kontextustól függően más-más jelentést vagy használatot rendelünk hozzá ugyanazon dologhoz – konkrétan, egy egyednek (egy változónak, függvénynek vagy objektumnak) többféle alakja létezhet. Az utóbbiból az következik, hogy többféle polimorfizmus létezik.

Ha egy függvény(neve)t túlterhelünk, azaz ugyanazon a néven más paraméterlistával definiáljuk, akkor végül is ugyanarra a névre más működést kapunk. Ha függvény sablont hozunk létre, akkor változó paramétertípussal hívhatjuk a függvényt. Tehát egy adott függvény végrehajtásának módja is változhat a paraméterek típusától függően. Ha ez egy kereső függvény és egész típusú változót kap paraméterként (például a dolgozók azonosítóival), akkor numerikus összehasonlítást, ha a paraméter szöveges típusú (a dolgozók névsorával) akkor szöveges összehasonlítást végez, vagyis ugyanaz a függvény a paraméter típusa alapján „dönti el", hogy hogyan működjön.

3.5. Elemi programszerkezetek

Tanulási célok

- Értelmezi és kiértékeli az input/output utasításokat.
- Értelmezi és kiértékeli a vezérlő utasításokat.
- Értelmezi és kiértékeli az aritmetikai és logikai műveleteket.

3.5.1. Input/output utasítások

A számítástechnikában, az input/output a számítógép és a külvilág közötti kommunikációra utal. (A külvilág a legtöbb esetben egy ember vagy egy másik információ feldolgozó rendszer.)

Az **input**, a rendszerbe bármilyen formában érkező adat lehet, a kimenet az az adat, amit kiküld. Az input és output utasítások nagyban különböznek szerkezetenként.

Az input vagy beolvasási parancs továbbítja az utasításokat, amik egy adott eszközön vagy porton keresztül érkeznek, - mint pl. a billentyűzet, az egér, a mikrofon – a memóriába, vagy egy speciális regiszterbe. Az **output**, vagy kimenet, írási parancs utasítást küld a memóriából vagy a regiszterből egy adott portra vagy eszközre, mint a képernyő, nyomtató vagy hangszórókészlet. Fontos megemlíteni, hogy a kimenet nem csak a monitor lehet, hanem egy file, egy adatcsatorna vagy a korábban említettek, amire át lehet irányítani a kimenetet. Ezek szabványos eszközök. Az adatfolyamot stream-nek is nevezik.

Az utasítások nyelvenként különböznek, de a lényegük, felhasználásuk ugyanaz. Ha a parancs nevében megjelenik egy f betű, akkor az a formázott be/kiviteli módra utal. Maga a parancs elég eltérő nyelvenként, sőt a Java kifejezetten macerás, ha a konzolról akarunk beolvasni, de nem erre találták ki.

Input utasítások

Ezek az utasítások beolvasnak, nézzük meg, hogy konzolról a különböző nyelvek hogyan juthatnak adathoz:

```
C#: Console.ReadLine()
Java: System.in.read() //ASCII kódot ad vissza
C++: cin >> valtozo
C: scan()
```

A C++ verzióban a szintaxis megköveteli, hogy a beolvasott értéket eltároljuk, a többi nyelven erről gondoskodni kell. Maga az utasítás nem hibás eltárolás nélkül, sőt használják is, tipikusan az utolsó utasításként, mintegy várva a felhasználótól bármilyen billentyű lenyomását, mielőtt tovább menne, vagy kilépne a program.

Output utasítások

Ezek az utasítások kiírnak valahova. Itt a konzolra való kiíratást nézzük nyelvenként:

```
C#: Console.WriteLine("szöveg")
Java: System.out.print("szöveg")
C++: cout << "szöveg"
C: print("szöveg")
```

3.5.2. A vezérlő utasítások

Ha egy számítógépen fut egy program, az végigmegy a kód első sorától az utolsóig. A vezérlő utasítások lehetővé teszik, hogy módosítsuk a számítógép vezérlését, automatikusan olvasva a következő kódsort, olvassa a másikat. A program sokkal hatékonyabb, ha irányítani tudjuk az utasítások sorrendjét.

A programok sorról sorra kerülnek értelmezésre. Ha szeretnénk, hogy bizonyos sorok kimaradjanak, vagy pont ellenkezőleg, sorokat szeretnénk még lefuttatni adott esemény bekövetkeztekor, esetleg többször szeretnénk végrehajtatni utasításokat, akkor az alább ismertetett konstrukciókat használhatjuk.

Minden kód C nyelven kerül bemutatásra, minimális módosítással átírható C++, Java, C# nyelvekre.

Az utasítások végrehajtását vezérlő szerkezetek a következők:

Az **if else** utasítás

A strukturált programozási szerkezetek közül ez az elágazás vagy szelekció. A lényege, hogy egy feltétel alapján bizonyos sorok kerülnek, vagy pont nem kerülnek végrehajtásra. A szerkezet kiértékelése után a program folytatja a futását.

Pszeudokóddal így fejeznénk ki:

```
ha (feltétel) akkor
        amikor igaz a feltétel
egyébként
        amikor nem igaz a feltétel
ezek a sorok már mindenképp értelmezésre kerülnek
```

konkrét felhasználása C nyelven:

```
int eredmeny; //így véletlenszerű érték lesz benne, de adhatunk értéket is: int eredmeny = 21;//többször nem deklarálhatunk
egy változót!
if (eredmeny >= 45)
        printf("Sikerult");
else
        printf("Nem sikerult");
```

A **switch/case** utasítás

Amennyiben egy változónk többféle értéket vehet fel (Fontos, hogy nem többféle feltételnek tesz eleget), akkor használjuk a switch/case szerkezetet. Megoldható if / else if / else szerkezetekkel is, de kevésbé átlátható.

Egy változó konkrét értékeit vizsgálhatjuk, pontosan megnézve, hogy felvett-e egy általunk előre definiált értéket. A változónak sorszámozott típusnak kell lennie (karakter vagy egész)! Vizsgálhatjuk azt az esetet is, amikor olyan értékkel rendelkezik, amit nem definiáltunk.

Egy jegyet (osztályzatot) szeretnénk szöveges formában megjeleníteni:

Pszeudokóddal így fejeznénk ki:

```
ha jegy = 0 akkor
        kiir („Nulla")
egyébként ha jegy = 1 akkor
        kiir („Elégtelen")
egyébként ha jegy = 2 akkor
        kiir („Megfelelt")
…
egyébként
        kiir („ez nem osztályzat")
```

Ha ezt if /else szerkezettel írnánk:

```
if (jegy == 0)
        printf("Nulla\n");
else if(jegy == 1)
        printf("Elégtelen\n");
else if(jegy == 2)
```

```
printf(“Megfelelt\n”);
…
else
        printf(“ez nem osztályzat\n”);
```

switch / case szerkezettel:

```
switch(jegy) {
        case 0 :
                printf(“Nulla\n”);
break;
        case 1 :
                printf(“Elégtelen\n”);
                break;
        case 2 :
                printf(“Megfelelt\n”);
                break;
        case 3 :
                printf(“Közepes\n”);
                break;
        case 4 :
                printf(“Jó\n”);
                break;
        case 5 :
                printf(“kiváló\n”);
                break;
        default :
 printf(“ez nem osztályzat\n”);
 break;
 }
```

A strukturált programozás másik szerkezete a szelekció mellett az **iteráció**. A szekvencia, mint utasítás az előző kettő szerkezet belsejében jelenik meg. Tehát nézzük a ciklusokat (iteráció).

Akkor használjuk őket, amikor ciklikusan szeretnénk utasításokat végrehajtani. A ciklusok fontos része a feltétel, ami eldönti, hogy hányszor hajtódik végre az utasítás(ok sorozata). A feltétel az úgynevezett ciklusfejben kerül meghatározásra, az utasítások pedig a ciklustörzsében, vagy blokkjában. Igen ritka az, amikor mindig ugyanazt az utasítást akarjuk többször felhasználni. Ezt is megtehetjük, de gyakoribb, hogy a ciklustörzsében állítjuk a feltételben vizsgált változó értékét, így érve el a feltétel hamis kiértékelését, és ezáltal a ciklus leállását. A ciklus leállta után folytatódik a kód értelmezése.

For ciklus – elöltesztelő, léptető ciklus

Felépítése:

```
for(értékadás; feltétel; léptetés) { ciklus törzse vagy a blokkja }
```

Az értékadás az “=” jellel történik, és értéket egy változónak tudunk adni. Pl.: i = 1

A feltétel a korábban megismert if szerkezet feltételével egyező szintaktikájú, pl.: i<15

A léptetés azt mondja meg, hogy ciklusonként mennyivel növeljük az értékadásban meghatározott változó értékét. Ezzel gyakorlatilag a leállás lehetőségét biztosítjuk. Pl.: i = i+1. Ez azt jelenti, hogy i értéke minden ciklusban 1-el növekszik. Egyszer csak eléri az 15-öt és akkor a feltétel – i<15 - hamissá válik, a ciklus leáll és folytatódhat a program futása a ciklus utáni kódokkal. Az eggyel való növelést C alapú nyelvekben így írják: i++

Egy for ciklus, ami kiírja a számokat 1 től 14-ig, C nyelven megvalósítva:

int i; //i –t deklarálnunk kell, hogy használhassuk a for-ban

```
for( i = 1; i < 15; i++) {
        printf("i erteke: %d\n",i);
}
printf("vege a for ciklusnak\n");
```

Az i értéke beállítódik 1-re, ez az értékadás. Ez a rész a ciklus futása során többet nem kerül értelmezésre. Megvizsgálja a feltételt, hogy igaz-e, ha igen, akkor végrehajtja az utasítást, ami jelen esetben i értékének kiírása. Aztán növeli i értékét eggyel, majd vizsgálja a feltételt, ha igaz, kiírja i értékét, ami most már 2. Amikor eléri a 14-et, a feltétel még igaz. Ki is írja, majd növel. 15-nél a feltétel hamissá válik, ezért ezt az értéket már nem írja ki. A ciklus utáni utasítások értelmezésre kerülnek a ciklus leállta után.

A kiíró utasítás printf parancs magyarázata:

A %d helyére egy decimális érték kerülhet, jelen esetben i konkrét értéke. Majd utána a \n az új sort jelenti.

Nézzük meg egy másik ciklussal ugyanezt a feladatot.

A **while** ciklus – elöltesztelő ciklus

Felépítése:

```
while(feltétel) { ciklus törzse vagy a blokkja }
```

Ha csak a felépítés szerinti formát írnánk be megfelelő szintaxissal, akkor egy úgy nevezett végtelen ciklust kapnánk. Ez azt jelenti, hogy nem tud a ciklus leállni, hiszen az előzőhöz képest, itt csak a feltételt adtuk meg, nincs leállást biztosító lépés.

Ezért a gyakorlatban az esetek túlnyomó részében az alábbi a felépítés:

```
értékadás
while(feltétel){ utasítás; léptetés }
```

Látható, hogy ugyanazokat a lépéseket kell felhasználni, mint a for ciklusnál, csak kicsit máshogy szervezzük a kódot.

Egy while ciklus, ami kiírja a számokat 1 től 14-ig, C nyelven megvalósítva:

FONTOS: értékadás a cikluson kívül!!!!

int i = 1; //i –t deklaráltuk, majd értéket is adtunk neki. egyszóval: inicializáltuk

```
while( i < 15 ) { //feltétel
        printf("i erteke: %d\n",i); //utasítás
        i++; //léptetés
}
printf("vege a while ciklusnak\n");
```

A kétféle ciklus közül néha az egyik jobban adja magát, mint a másik, de alapvetően mindenki azt használja, ami szimpatikusabb. Esetleg mondhatnánk: amikor tudjuk, hogy pontosan 14-szer kell lefutnia a ciklusnak, akkor a for-t használjuk. Ha nem tudjuk, hogy pontosan hányszor fog lefutni, akkor a while-t. De mindkettővel megoldhatóak a feladatok.

Az utolsó ciklusfajtánk:

A **do / while –** hátultesztelő ciklus

Felépítése:

```
do{ ciklustörzse} while(feltétel);
```

Ugyanaz igaz rá mint a while-ra, abból a szempontból, hogy itt is a cikluson kívül szokták inicializálni a változót, amit a blokkban léptetnek. A lényegi különbség, hogy a ciklus futásának feltételét a végén ellenőrzi, ezzel azt érve el, hogy egyszer mindenképpen lefut a ciklus. Ez ritkábban használt forma. Tipikus felhasználási területe, ha bekérünk egy adatot s utána azt ellenőrizzük. Nem tudjuk, hányszor kell bekérni az adatot, míg az ellenőrzés után el tudjuk fogadni, de egyszer biztosan le kell futnia, mert az adatbekérésre szükség van.

Kérjünk be 1 számot a felhasználótól, ami 15-nél kisebb. C nyelvű megvalósítása:

```
int szam;
do{
 printf(„Kerek egy szamot, ami < 15 :”);
 scanf(„%d”,&szam);
}while(szam >= 15);
```

A közvetlen vezérlésátadó utasításokat strukturált programban nem szabad használni. Azonban a break és continue parancsok megengedettek, bár elkerülhető a használatuk. A return, mint egyetlen kilépési pont használható, sőt, kötelező eleme bizonyos szerkezeteknek (függvény visszatérési érték megadása).

A **goto** utasítás

Feltétel nélküli vezérlés átadás. Kompatibilitási okokból a nyelvek felismerik a goto parancsot, azonban nem javasolt használni. Itt gyakorlatilag arról van szó, hogy a vezérlést egy másik sorra irányítjuk. A parancs használatával össze-vissza lehet ugrálni a kódban, ezáltal átláthatatlan, strukturálatlan kódot kapva.

A goto utasítás formája:

```
goto cimke

…
cimke : utasítások
```

Kerüljük!!!

A break utasítás

Az aktuális ciklus elhagyására szolgál, egymásba ágyazott ciklusok esetén csak az aktuális szintről lép ki., amennyiben egy bizonyos feltétel bekövetkeztekor abba akarjuk hagyni a ciklus futását. Az előbbi példáknál maradva, a ciklusunk 13-nál hagyja abba a futást:

```
int i;
 for(i = 1; i < 15; i++) {
 printf(„i erteke: %d\n”,i);
 if( i == 13) break; // e nélkül a 14-et is kiírná
 }
```

continue utasítás

Az aktuális iterációt eggyel lépteti, tehát erről a sorról ugrik a program a ciklus fejrészére, mintha rendesen lefutott volna a ciklus, de nem hajtja végre a continue utáni részt.

Most írassuk ki a számokat 1-14-ig, de a 13 ne íródjon ki:

```
int i;
 for(i = 1; i < 15; i++) {
 if( i == 13) continue;//ha igaz a feltétel, erről a sorról a fejre ugrik a vezérlés, ott növelődik i
 printf(„i erteke: %d\n”,i);// a 13 –at nem írja ki
 }
 printf(„vege a ciklusnak\n”);
```

return utasítás

Ez az utasítás a függvények végén található. A függvény ad visszatérési értéket, ebben különbözik az eljárástól. Ha kiszámolt valamit a függvényünk, akkor az eredményt a return utáni értékként adja vissza.

Felhasználására példák:

```
return "ok"; //konkrét szöveget
return 21; //konkrét számot
return eredmeny; //bármilyen változót
```

Egy összeadást végző függvény visszaadja a kiszámolt eredményt, ami csak egész érték lehet:

```
int iOsszead(int szam1, int szam2){
 return szam1 + szam2;
}
```

Egy összeadást végző függvény visszaadja a kiszámolt eredményt, ami valós érték lehet:

```
double dOsszead(double szam1, double szam2){
 return szam1 + szam2;
}
```

Egy függvény visszatérési értékének felhasználása:

```
int main()
{
 int egyikSzam = 21;
 double eredmeny;
 eredmeny = dOsszead(egyikSzam, 3.14);
 printf(„Az osszeadas eredmeny: %.2f\n",eredmeny);
}
```

A printf függvényben a %.2f helyére egy valós érték kerülhet, 2 tizedesjegy pontossággal. Most az eredmeny változó értéke fog kiíródni.

3.5.3. Aritmetikai és logikai műveletek

Az aritmetikai műveleteket számokkal végezzük és számokat kapunk eredményül.

Az aritmetikai műveletek közé tartozik az összeadás, a kivonás, a szorzás és az osztás. Minden programozási nyelven elérhetőek ezek a műveletek, sőt, gyökvonást, hatványozást, illetve további műveleteket is támogatnak a ma használatos nyelvek. (lásd 1. melléklet)

+	összeadás
-	kivonás
*	szorzás
/	osztás
%	maradékos osztás

Az összeadás, kivonás, szorzás és osztás műveletek megfelelnek a hozzátartozó matematikai műveleteknek. Ha el akarjuk tárolni az eredményt , akkor az „=" jel bal oldalán van a tárolásra létrehozott változó, jobb oldalon pedig a matematikai kifejezés.

A maradékos osztás az a művelet, amely megadja két érték elosztásának fennmaradó értékét.

Például, ha azt írjuk: A = 13 % 3;

A változó 1 értéket fogja tartalmazni, mert a 13 és 3 közötti osztásban a fennmaradó rész 1.

Művelet	Leírás	Kód Pl.
*	Szorzás	e=a*b
/	Osztás	f=a/b
%	Maradék (csak egész számoknál)	g=a%b;
+	Összeadás	c=a+b;
-	Kivonás	d=a-b;

Összehasonlító műveletek: azonos típusokat hasonlíthatunk össze, eredményül logikai értéket kapunk.

Az „a" és „b" egyforma típussal rendelkező (mondjuk egészek) változók, az éppen aktuális értékeik kerülnek összehasonlításra, minden kiértékelés igaz vagy hamis eredménnyel zárulhat.

Művelet	A feltétel igaz, ha:	Kód Pl.
==	Az értékek egyenlők	a==b
!=	Az értékek nem egyenlők	a!=b
>	„a" nagyobb, mint „b"	a>b
<	„a" kisebb, mint „b"	a<b
>=	„a"nagyobb vagy egyenlő, mint „b"	a>=b
<=	„a"kisebb vagy egyenlő, mint „b"	a<=b

A logikai műveletek logikai értékeket hasonlítanak össze, logikai értéket adva eredményül. A logikai érték igaz vagy hamis lehet. Típusként bool vagy boolean szavakkal deklaráljuk. Értéke, akár csak a deklaráció, nyelvfüggő. true/false vagy 1/0 lehet.

Logikai műveletek közé tartozik az AND, az OR és a NOT, ezeket arra használjuk, hogy összehasonlítsanak két logikai kifejezést, és visszaadjanak egy logikai eredményt. (A NOT nem hasonlít össze két értéket)

Amikor több feltételt akarunk összekapcsolni, elkerülhetetlen a használatuk. Ha létezik kettő (vagy több) feltételünk: a<b, c>d, és e kettő feltételt egyszerre szeretnénk használni, akkor nem mindegy, hogyan kapcsoljuk össze őket. A feltételek egyenként igaz vagy hamis eredményt adhatnak. A kérdés, hogy mit is szeretnénk? Az összekapcsolt feltételek mindegyikének teljesülnie kell, vagy elég csak az egyiknek?

Ha ÉS (AND, &) kapcsolatba hozzuk a feltételeket, akkor mindegyik feltételnek teljesülnie kell. Ha VAGY (OR, |) kapcsolatot alakítunk ki, akkor elég az egyik feltételnek igaznak lennie.

A jelét a "w" billentyűzeten találjuk.

Az alábbi táblázat mutat egy összefoglalót:

Szimbólum	Betűvel	Definíció
&	AND	Ha minden feltétel igaz, akkor igazat ad vissza, különben hamisat
|	OR	Ha legalább egy feltétel igaz, akkor igazat ad vissza, különben hamisat
~, !	NOT	Megfordítja az eredményét a relációs műveletnek

A C alapú nyelvekben többnyire a szimbólumokat használjuk, abból is kettőt, egymás mellet.

Pl.: (a < b) && (c> d)

A zárójelezés jelen esetben csak a jobb olvashatóság miatt használatos. Ez a feltétel azt jelenti, hogy a legyen kisebb, mint b, és c legyen nagyobb, mint d. Az ÉS kapcsolatnak köszönhetően mindkét feltételnek teljesülnie kell.

Gondoljunk a felhasználó név, jelszó párosra. A kettő helyességének egyszerre kell teljesülnie, hogy beengedjenek:

```
if( felNev == „RokaRudi” && jelszo == 123 )
        printf („Rendben”)
else
        printf(„HIBA”)
```

pl.: (a < b) || (c> d)

Ez a kifejezés azt jelenti, hogy elég az egyik feltételnek teljesülnie. A fordító először megnézi, hogy a kisebb-e mint b. Ha igen, nem is foglalkozik a másik oldallal, hiszen elég, ha a kifejezés egyik tagja igaz. Amennyiben hamis, akkor megvizsgálja a másik tagot is. Ha igaz, akkor az egész kifejezést igaznak minősíti.

Itt gondoljunk arra, hogy egy regisztráció során a jelszó hosszát ellenőrizzük:

```
if(hossz < 3 || hossz > 8)
 printf(“Nem jo formatum!”);
else
 printf(“Elfogadva”);
```

Akármelyik feltétel teljesül, nem fogadhatjuk el a jelszót. Itt fordítva gondolkodtunk, az a jó, ha hamisat ad az összekapcsolt feltételünk.

Ugyanez a feltétel && jellel összekapcsolva mindig hamisat adna, azaz mindent elfogadna. Ugyanis nincs olyan szám, ami kisebb, mint 3 ÉS nagyobb, mint 8.

Az előbbi fordított logikát egészítsük ki a következő logikai művelettel, a negálással vagy tagadással. NOT vagy ! a kódban használatos jelölése, jelentése pedig, hogy tagadja a feltételt:hamisból igazat, igazból hamisat csinál.

Vizsgáljuk, hogy egy ellenőrzés során bármilyen hiba fellépett-e. A hiba megjelenését egy logikai változó tartalmazza, kezdetben false értékkel, ha valahol hiba van, akkor ott átbillentjük a kapcsolót true értékre. Később az értékét fogjuk vizsgálni:

```
bool vanHiba = false; //inicializálás, feltételezzük, hogy nincs hiba
// vizsgálódás, mondjuk nem megfelelő a jelszó hossza
if(hossz < 3 || hossz > 8)
 vanHiba = true;
//további vizsgálódások
//ha nincs hiba, akkor sem állítjuk sehol false-ra sehol!!!
//hiba változó értékének függvényében viselkedik a továbbiakban a program:
if( vanHiba == false )
        printf(„minden rendben”);
else
        printf(„Hiba lépett fel, próbálkozzon később!”);
```

Hát, ebben nincs negálás, de a van Hiba változó másként is vizsgálható:

```
if (!vanHiba) // ez ugyanazt jelenti, mint if( vanHiba == false )
```

A másik felhasználás, amikor tényleg negálunk.

Képzeljük el, hogy van egy darab kapcsoló, ami fel- és lekapcsolja a villanyt. Ezt 1 kapcsolóval akarjuk megoldani, mondjuk, hogy a bool villany = true azt jelenti, hogy ég a villany. Ha rákattintunk, akkor villany = false, nem ég a villany. Ha most kattintunk, akkor megint villany = true, s világos van.

Ezt if szerkezettel megoldva:

```
if(villany)//ugyanaz mint: if (villany == true)
        villany = false
if (!villany) //ugyanaz mint: if (villany == false)
        villany = true
```

De mondhatnánk egyszerűen azt is, hogy:
villany = !villany

A sor olvasása: villany legyen egyenlő az állapotának a negáltjával. Ha true volt, legyen false, és fordítva.

Vigyázzunk, kicsit elírva megint zagyvaságot kapnánk, de a program nem jelezné, szintaktikailag rendben van, „csak" szemantikailag hibás a következő sor:
villany != villany
villany != !villany

1. Jelentése: villany nem egyenlő saját magával. Ez mindig hamis.

2. Jelentése: villany nem egyenlő a negáltjával. Ez mindig igaz. A true nem egyenlő negáltjával, azaz a false-al.

Azokat a jeleket, amiket itt használtunk, operátornak hívják. Ezek a relációs jelek (<,>,=), a logikai jelek (&,|,!) és az aritmetikai jelek (+,-,/,*,%). Ha két érték között áll, akkor két operandusú az operátor, egyébként egyoperandusú. A !, mint negáló operátor egyoperandusú, hiszen egy értéket változtat meg. Ha != (nem egyenlő) operátort nézzük, akkor ez két operandusú, mert 2 feltételt hasonlít össze.

3.6. Tesztelés

Tanulási célok

- Meghatároz alapvető tesztelési fogalmakat, mint error, fault, failure. Felismeri az ellenőrzés, a tesztelés és a hibakeresés különböző szintjeit.
- Bemutatja az egységteszt, a rendszerteszt és az átvételi teszt eltérő céljait és alkalmazási területét.
- Különbséget tesz statikus és dinamikus tesztelési módszerek között, és példákat sorol fel automatikus tesztelő eszközökre.

3.6.1. Alapvető tesztelési fogalmak, az ellenőrzés, a tesztelés és a hibakeresés különböző szintjei

A szoftvertesztelést le lehet írni, mint egy érvényesítési folyamat, mely során ellenőrzik, hogy a szoftver megfelel-e az előírt követelményeknek, és egyúttal segít a hibák feltárásában is. A szoftvertesztelésnek három fő célja van: a hitelesítés, az érvényesítés és a hibák felfedezése.

A hitelesítés az a folyamat, amely ellenőrzi, hogy a program megfelel-e a műszaki előírásoknak. Hibamentes-e?

Az érvényesítés az a folyamat, mely során azt ellenőrizzük, hogy a szoftver megfelel-e az üzleti követelményeknek. A hiba felfedezésének az a célja, hogy megtalálják és kijavítsák a hibákat mind a programkódban, mind a megjelenésben.

Fontos ellenőrizni, hogy a program nemcsak helyesen, de az elvárásoknak megfelelően is működik-e.

A compileres nyelvekben addig nem fordul le a program, amíg szintaktikai hibákat vagy gépelési hibákat

talál a fordító. Azonban minden program tesztelésének célja a futásidejű hibák feltárása, amelyek a következőek: kimeneti hibák és kivételek.

A kivétel szó helyett nem véletlen, hogy nem a hiba szó szerepel. A kivétel egy nem várt fordulat, de mindenképpen kezelnie kell a váratlan fordulatokat is a programnak. Pl. nem elérhető az adatbázis.

A hibakeresés a futásidejű hibák javításának folyamata. A hibakeresést legtöbbször kis léptékekben végzik, így a hibák menet közben javíthatóak. Így sokkal könnyebb a hibákat megtalálni, mint később egy nagy programban.

3.6.2. Az egységteszt, a rendszerteszt és az átvételi teszt

Az **egységtesztelésben** a tesztelést annak az érdekében végzik, hogy ellenőrizzék, hogy egy adott modul vagy egység kódja jól működik-e. Egység lehet osztály is. A moduláris programozás szellemében a kódokat külön szervezzük, ha objektumorientáltan programozunk, akkor osztályokba. Egy vagy több osztály alkotja a modult. Ha strukturáltan programozunk, akkor külön fájlokba rakjuk az egyfajta feladatot ellátó függvényeket, eljárásokat. Az egységtesztet nagyon alapvető szinten végzik, amikor az egység kódját kidolgozták, vagy egy adott funkciót beépítenek. Az egységteszt az egység teljes vizsgálatával foglalkozik. Ellenőrizni kell, hogy egy metódus önmagában megfelelően működik-e, majd egy modulteszt keretében az együttműködést is. Általános értelemben, a „rendszer tesztelése" kifejezés arra utal, hogy a rendszert tesztelik, mesterséges körülmények között, annak érdekében, hogy a vártnak és a szükségeknek megfelelően működjön. Ez egy átfogó vizsgálat a teljesen összeállított szoftvertermékén.

A modulok átmentek a teszteken, erre szolgált az egységteszt, most a rendszer egésze kerül tesztelésre.

Rendszer tesztelése nem foglalkozik a rendszer funkcionalitásával, és azzal, hogy megfelel-e a felhasználók igényeinek.

Az **átvételi tesztben** a szoftvert átadják a felhasználóknak, annak érdekében, hogy megbizonyosodjanak arról, hogy a szoftver megfelel-e a felhasználói elvárásoknak. A szoftverfejlesztésnek ez egy olyan szakasza, ahol a szoftver tesztelése éles környezetben, a célközönség által kerül tesztelésre.

Piaci szoftver esetében gyakori az úgynevezett béta teszt, amikor a célközönség szűk rétege éles körülmények között használja a szoftvert, és küldi a visszajelzéseket.

Egyedi alkalmazások esetén az alfatesztet a megrendelő végzi, kipróbáláskor, betanításkor.

Bármilyen teszt során az eredményeket dokumentálni kell. Milyen adatokra, hogyan reagált a program. Természetesen a felmerülő hibákat javítani kell.

3.6.3. Statikus és dinamikus tesztelési módszerek, automatikus tesztelő eszközök

A statikus elemzés során a kódot „nézegetjük", dinamikus tesztelés során futtatjuk is a programot.

A **statikus tesztelés** során elsődlegesen a kódot értelmezzük sorról sorra. Magyarázhatjuk magunknak is, meglepően hatásos, de még jobb, ha valaki másnak mondjuk el, hogy mit is csinál a program. Általában nem egy részletes vizsgálat, főként a kód, algoritmus, dokumentum ésszerűségét nézi. Elsősorban a kód szintaktikai ellenőrzése, a kód manuális olvasása, vagy a dokumentumban lévő hibák keresése. A statikus elemzési technikák a lefordítás előtti forráskód vizsgálata köré épülnek. Kiindulásképpen vizsgálják az alapvető utasításkészletet a nyers formában és próbálják megtalálni a szemantikai és logikai hibákat benne.

Dinamikus tesztelés során használjuk a programot. Itt további kettő technikáról beszélünk: feketedoboz tesztelés, fehérdoboz tesztelés.

Feketedobozos teszteléskor nem használjuk a forráskódot, a működő programot úgy vizsgáljuk, hogy a bemeneti adatokra megfelelő eredményt kaptunk-e.

Fehérdobozos tesztelésnél a tesztadatokat úgy választjuk meg, hogy ismerjük a kód felépítését is.

Természetesen minden módszernél fontos a megfelelő tesztadat kiválasztása.

A 2+2 bemenetre jó a 4, mint eredmény, de nem mindegy, hogy összeadta, vagy összeszorozta a számokat a program.

Tesztelés	
Statikus	Dinamikus
Kódellenőrzés • formai • tartalmi – felhasználatlan változó – felhasználatlan objektum – érték nélküli változó – végtelen ciklus	Feketedoboz • ekvivalencia osztályok • határesetek Fehérdoboz • kipróbálási stratégiák • különböző tesztek – modul teszt – biztonsági teszt – összeépítési teszt stb.

A különböző fejlesztői környezetekben van lehetőség automatizált tesztelésre is. Ezt tesztvezérelt fejlesztésnek hívják, lényegében arról van szó, hogy teszt programot írunk, ami vizsgálja az éles programunkat. Java nyelvhez Junit, C# nyelvhez NUnit a legismertebb tesztkörnyezetek.

Az automatizált tesztelési eszközök további lehetőségei a következők:

Rational Robot az IBM-től
A Rational Robot egy teszt-automatizálási eszköz kliens/szerver alkalmazások funkcionális tesztelésére. Lehetővé teszi, hogy a teszt-automatizálási mérnökök, teszt szkriptek kiterjesztésével felismerjék a hibákat, és meghatározzák a teszteseteket. Ez vizsgálati eseteket biztosít az általános tárgyak számára és speciális teszteket a fejlesztői környezet tárgyainak. Segíti a hibakövetést, a változások kezelését és a követelmények nyomon követhetőségét. Ez magában foglalja a beépített teszt-menedzsmentet és más IBM Process eszközökkel való integrálását.

QuickTest and Winrunner Mercurytől
A **Winrunner** segít automatizálni a tesztelési folyamatot, a tesztfejlesztéstől a kivitelezésig. Rugalmas és újrafelhasználható teszt szkripteket hoz létre, amelyek az alkalmazások funkcionalitását teszik próbára. A szoftverkiadását megelőzően, egyik napról a másikra le lehet futtatni ezeket a vizsgálatokat, amely lehetővé teszi a hibák észlelését és biztosítja a kiváló szoftverminőséget.

A **QuickTest** egy automatizált funkcionális grafikus felhasználói felületet (GUI) tesztelő eszköz, amely lehetővé teszi a felhasználói tevékenységek automatizálását weben, vagy kliens alapú és asztali számítógépes környezet felhasználásával. Ezt elsősorban funkcionális regressziós tesztautomatizálásra használják.

A regressziós tesztelést (más néven ellenőrző tesztelést) azután kezdeményeznek, miután a programozó megpróbálta kijavítani a felismert problémákat, vagy hozzáadott a programhoz egy forráskódot, ami esetleg meghibásodhatott emiatt. Ez egy minőségellenőrzési intézkedés, amely azt vizsgálja, hogy az újonnan módosított kód továbbra is megfelel-e az előírt követelményeknek, és, hogy a módosítatlan kódrészt nem érinti az elvégzett karbantartási tevékenység.

TestPartner és QARun a Compuware-től
A **TestPartner** egy olyan eszköz, amely lehetővé teszi a felhasználók számára, hogy létrehozzanak automatizált teszt szkripteket, az alkalmazások működésének érvényesítésére. Ez webes, és ügyfél/kiszolgáló típusú alkalmazások tesztelésére használható. Az egyik előnye az, hogy rendelkezik beépített funkcióval, amely lehetővé teszi a felhasználó számára az SAP rendszerek tesztelését. Végre tud hajtani olyan vizsgálatokat, amelyek QARun használatával lettek létrehozva, a Compuware régebbi funkcionális automatizált tesztelő eszközével.

A Testpartner szkriptei Microsoft Visual Basic for Applications (VBA) segítségével jöttek létre, és mivel ez egy nem védjegyezett nyelv, így meglehetősen nagy a felhasználói bázisa az automatizált szoftvertesztelés területén kívül is. Segítséget nyújt mind a hibakereséssel, mind a futásidejű változókkal kapcsolatos

információival. Ezenkívül varázslók is rendelkezésre állnak, hogy segítsenek olyan részek érvényességének ellenőrzését létrehozni, mint a tárgyak tulajdonságai és a rendszer adatai.

A **QARun** egy vállalati szintű teszt szkript, fejlesztési és végrehajtási eszköz, a Compuware's QACenter alkalmazás tesztelési termékcsalád részeként. Az automatizált képességek javítják a tesztelők termelékenységét, miközben képessé teszik őket arra, hogy hatékonyan hozzanak létre teszt szkripteket, teszteket ellenőrizzenek, és teszteredményeket elemezzenek. A QARun segítségével, a tesztek könnyen újra felhasználhatók.

Axe az Odin Technology-tól
Az **Axe** egy teszt automatizálási platform, amely nagyban növeli a termelékenységet, és jelentősen csökkenti a teszt automatizálás karbantartási költségeit. Eszközként egy egyszerű Microsoft Excel táblázatkezelőt használ, hogy meghatározza és modellezze a teszt forgatókönyveket és kódgenerálási technikákat, és így gyorsan generáljon robosztus, önállóan dokumentált automatizálási kódot.

3.7. Dokumentálás és karbantartás

Tanulási célok
- Bemutatja a szoftver fejlesztése és átadása során használatos dokumentációkat, mint a mondatszerű leírás, a döntési fák, az UML kód, a kódban elhelyezett kommentek és a folyamatábrák.
- Bemutatja a jól strukturált és jól dokumentált programkód előnyeit.
- Vázolja, hogy hogyan kell dokumentálni a módosításokat a szoftverben és a program dokumentációjában.
- Bemutatja azokat a módszereket, amelyek biztosítják a programkarbantartás minőségét, mint a kódfelügyelet, a kód kommentezésének szabályai és a technikai referencia-kézikönyvek.

A szoftverfejlesztéshez kapcsolódó dokumentációk

Mondatszerű leírás

A mondatszerű leírás, mint a neve is sugallja, egyfajta szöveges leírás,

A mondatszerű leírás esetén a beszélt nyelv segítségével fogalmazzuk meg a problémát, a strukturált szerkezetek használatával, de a konkrét megvalósítással nem foglalkozva. Pszeudokódnak is nevezik.

```
PROGRAM
        CIKLUS
BE: a, b
        CIKLUS VÉGE amíg a < 0 VAGY b < 0
        csere(a,b)
        KI: a
PROGRAM VÉGE
```

A fenti pszeudokód implementálásakor (konkrét nyelven való megvalósításakor) szükségünk lesz egy hátultesztelő ciklusra, amiben a beolvasást meg kell valósítani. A ciklusból akkor lépünk ki, ha mindkét beolvasott érték nagyobb, mint 0. A cseremetódus konkrét megvalósításával nem foglalkozunk. Természetesen egy másik pszeudokódban vagy folyamatábrában kifejtésre kell kerülnie.

A szoftverek dokumentálásában a mondatszerű leírás használatának előnyei az alábbiak:
- Egyértelművé teszi a logikát és a kapcsolatokat, amik a használt nyelvben megtalálhatók.
- Ez egy hatékony kommunikációs eszköz, amit könnyű tanítani és megérteni.
- Strukturált, ellentétben a döntési táblával, és a döntési fákkal, amelyek csak az elágazások logikáját mutatják.
- A mondatszerű leírás a teljes utasítást tartalmazza, lépésről-lépésre.
- Nyelvfüggetlen.

A döntési táblák egy lehetőséget nyújtanak arra, hogy megvizsgáljanak, leírjanak vagy dokumentáljanak döntéseket táblázat segítségével.

Az alábbiakat használják:

- Feltételek megadása
- A lehetséges döntési lehetőségek meghatározása
- A végrehajtandó műveletek
- Műveletek leírása

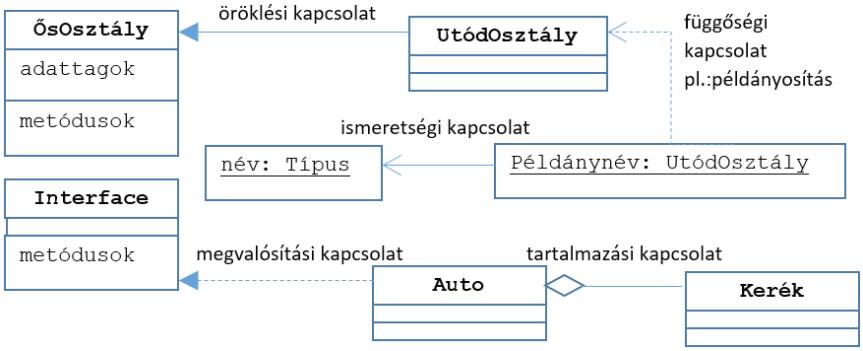
Döntési fák

Döntési fákat használnak, amikor összetett elágazás fordul elő egy strukturált döntési folyamatban, tehát több lehetőségből kell választani. A fák akkor is hasznosak, ha elengedhetetlen fontosságú megtartani a döntéseket egy adott sorrendben.

Matematikai értelemben a döntési fa gráf.

Az Unified Modeling Language

Az Unified Modeling Language (UML), egy szabványos nyelvet biztosít a szoftver rendszerek összetevőinek megjelenítésére, felépítésére és dokumentálására. Az UML magában foglal egy sor grafikus jelölési technikát, egyedi rendszerek absztrakt modelljeinek létrehozására. Ez magában foglalja az elvi elemek, mint a szereplők, az üzleti folyamatok, rendszerösszetevők és a rendszer tevékenységét. Ugyancsak ide tartoznak a konkrét dolgok, mint újrafelhasználható szoftver elemek, adatbázissémák és programozási nyelv nyilatkozatok.

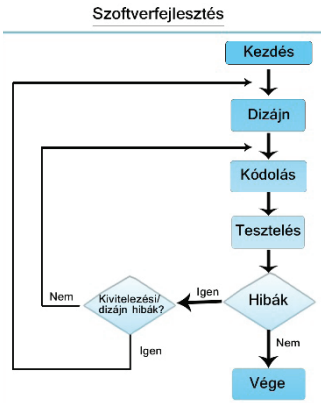


3.48. ábra A legfontosabb UML jelölések

Kódban elhelyezett kommentek

A kódban elhelyezett kommenteket abból a célból adja a programozó a programkódhoz, hogy könnyebb legyen megérteni a kódot. A szintaxis és a kommentelési szabályok a programnyelv specifikációjában vannak meghatározva.

Folyamatábrák



3.49. ábra A szoftverfejlesztés folyamatábrája

A folyamatábrákat általában egy üzleti folyamat vagy üzleti szabály logikájának részletesebb leírására használják. A folyamatábrák a fizikai információáramlás grafikus illusztrációi, a teljes üzleti rendszerben. A folyamatábrákat rendszeresen használják elemzésre és tervezésre. A vonalak jelképezik a folyamatok sorozatát, és egyéb szimbólumok jelzik a be- és kimeneteket a folyamatban. Számítógépes folyamatok, manuális műveletek, műveleti rendszerek ki- és bemenetei írhatók le vele. Ezeket arra tudják használni, hogy fő ellenőrző pontokat azonosítsanak egy rendszer belső ellenőrző szerkezetében. A folyamatábra megmutatja a szoftverfejlesztés folyamatát.

Egy algoritmus magyarázatára pszeudokódot, struktogramot vagy folyamatábrát szoktak használni. Az osztályok kapcsolatát UML-ábrával szemléltetik.

3.7.1. A jól strukturált és jól dokumentált programkód előnyei

A strukturált kód előnyei közül néhány:

- A kód modularitását növeli, hogy a helyi változások nem terjednek ki az egész programra.
- A program karbantartása és átalakítása egyszerűbb, ha a struktúra egyszerű és szabályos.
- Könnyebb megérteni a program logikáját.
- A legtöbb teljesítménybeli probléma az alkalmazásban csak a kód megváltoztatásával, vagy hozzáadásával oldható meg, ha a kód jól felépített és a modulok lazán összekapcsoltak, akkor sokkal egyszerűbb a módosítás.

A kód dokumentálásának célja, hogy segítse a megértést. Könnyű elfelejteni, hogy miért csináltál valamit, és hogy hogyan. Könnyen összekeverhetők a hasonló program verziókkal.

A jól dokumentált kód jól elnevezett szabályokat, logikai modularizációt, egyszerű technikákat használ, minden feltételezés jól dokumentált benne és rendszeresek a jó kommentek.

Célszerű szem előtt tartani azt is, hogy a kód minél jobban magyarázza magát. Megfelelő konvenciók betartásával, tervezéssel elérhető ez a fázis.

3.7.2. A módosítások dokumentálása

Változások történhetnek a szoftverben a program különféle okai miatt. Azok a változások, amiket azért készítettek, hogy kijavítsák a hibákat a szoftverekben, korrektív karbantartások. Azok a változások, amiket azért csinálnak, hogy javítsák a program teljesítményét, fenntarthatóságát vagy a szoftver más attribútumait, a rendszer perfektív karbantartása. Ha a szoftverváltoztatás azért történik, hogy használhatóvá tegye a szoftverrendszert egy megváltoztatott környezetben, adaptív karbantartásról beszélünk.

Amikor a szoftverrendszerben változás történik, a kezdeti fejlesztés alatt, vagy a kibocsájtás utáni karbantartás során, regressziós tesztet végeznek, hogy a nem módosított területek bizonyíthatóan továbbra is jól működnek-e a változtatások után is. Tesztelni az tud, aki nem vett részt a fejlesztésben, ezért a pontos és részletes dokumentáció elengedhetetlen.

Általában érvényben lesz egy **Software Document Control Procedure** (Szoftver Dokumentum Ellenőrző Eljárás), amely meghatározza a módszereket, és az az iránti kötelezettségeket, továbbá azt, hogy hogyan irányítsák a dokumentumok felülvizsgálatát, jóváhagyását, terjesztését, és biztosítsák a szoftver referenciákat és képzési anyagokat. Ez azért fontos, mert a látszólag jelentéktelen változtatások a szoftver kódjában váratlan és nagyon jelentős problémákat okozhatnak máshol a programban. A szoftverfejlesztési folyamatnak jól megtervezettnek, ellenőrzöttnek és dokumentáltnak kell lennie, hogy feltárja és korrigálja a nem várt eredményeket a szoftver változtatásakor.

A jó változásmenedzsment rendszer használni fog értesítéseket és jóváhagyási folyamatokat, mielőtt bármilyen változtatás történne. Ez gyakran bekövetkezik, így a munkafolyamat során érintett felhasználók, vagy a rendszer tulajdonosai értesítéseket kapnak a sorozatosan vagy párhuzamosan tervezett változásokról. A módosítást kérő személy részletesen indokolja a változtatás okát, a változás várt eredményeit, helyreállítási tervet, arra az esetre, ha a változtatás előre nem látható következményekkel járhat.

Dokumentációt gondosan meg kell vizsgálni abból a szempontból, hogy mely dokumentumokat érinti a változás. Minden jóváhagyott dokumentumot, azaz előírást, vizsgálati eljárást, felhasználói kézikönyveket, stb., amiket érintenek, a konfiguráció kezelési eljárásokkal naprakésszé kell tenni. Előírásokat naprakésszé kell tenni, mielőtt bármilyen karbantartási és szoftver változás történne.

3.7.3. A programkarbantartás minőségbiztosítása, a kód kommentezésének szabályai

Amikor korrekciós karbantartási programokat végzünk egy szoftver programon, célunk a teljesítmény javítása. Ennek egyik módja a formális kód vizsgálata. A formális kód ellenőrzése az egyik leghatékonyabb rendelkezésre álló technika a kód minőségének javítására.

A **kódfelügyelet** általában más jellegű hibákat talál, mint amit a teszteléssel feltárnak. Ebből kifolyólag kódellenőrzést gyakran végeznek a tesztelések megkezdése előtt, ezáltal nagymértékben csökkentve a rendszer hibáit. A kódfelügyeleti munka akkor a legjobb, amikor a végrehajtó csapat a következőkből áll:

- Moderátor, akinek a feladata, hogy megszervezze a találkozókat, vezeti a vizsgálati folyamatot, és nyomon követ minden utómunkát.
- Egy olvasó, aki átveszi a kód működését a csapattal, úgy, hogy saját szavaival körülírja azt.
- Egy jegyző, aki egy előre meghatározott szabványos űrlapon minden hibát figyelembe vesz, ezzel felszabadítva a csapat tagjait, hogy csak a kódra összpontosítsanak.
- A kód írója, aki megért minden hibát, amit találnak, és elmagyaráz minden nem egyértelmű részt.

A jó **kód komment** leírja a kód szándékát. A függvényekben vagy eljárásokban lévő kód célját kell leírnia, persze a változókról is érdemes hasznos információkat rögzíteni. Minden kommentnek pontosnak és szükségszerűnek kell lennie. A megfogalmazásnak egyértelműnek kell lennie. A kommentek lehetnek egysorosak, vagy alkothatnak kommentblokkot is. Mivel a megjegyzések nem a kód futtatásához tartozó részek, külön jelölést kapnak mind a fordító, értelmező, mind a programozó számára.

Ha egy alkalmazást megvalósítottak és a vevő elfogadta, a szükséges dokumentációt csatolni kell szoftverhez.

Ez a **dokumentáció** tartalmazza a felhasználói kézikönyvet, a műszaki paramétereket és a technikai referencia-kézikönyvet. A technikai referencia-kézikönyv azoknak az embereknek készül, akik támogatni fogják a rendszert, és kezelik a rendszerrel kapcsolatban felmerülő felhasználói kérdéseket.

Erre azoknak az embereknek van szükségük, akik a szoftvert fejlesztési célokra használják. A technikai referencia-kézikönyvben leírják a parancsokat, függvényeket, modulokat, és az alkalmazások beállításait.

3.8. Programozási példák

Tanulási célok

- Értelmez egy adott hipotézis alapján készített kisebb programrészletet.
- Azonosítja a kód hibáit vagy gyengeségeit, és elvégzi a követelményeknek megfelelő módosításokat.

3.8.1. Példaprogramok

A program olyan parancsok gyűjteménye, amelyek megmondják a számítógépnek, hogy csináljon „valamit”. A parancsoknak ezt a gyűjteményét nevezik kódnak. Mi az EPL-t (Eucip Programming Language) fogjuk használni, ami egy egyszerűsített változata a C programozási nyelvnek. (Lásd 1. melléklet)

Példa 1

A legminimálisabb program, ami ugyan nem csinál semmit, de minden kódban megjelenik:

```
main(){}
```

Ha ki szeretnénk íratni valamit a képernyőre, a korábban ismertetett printf parancs segítségével, akkor a következő sorokat kell begépelni:

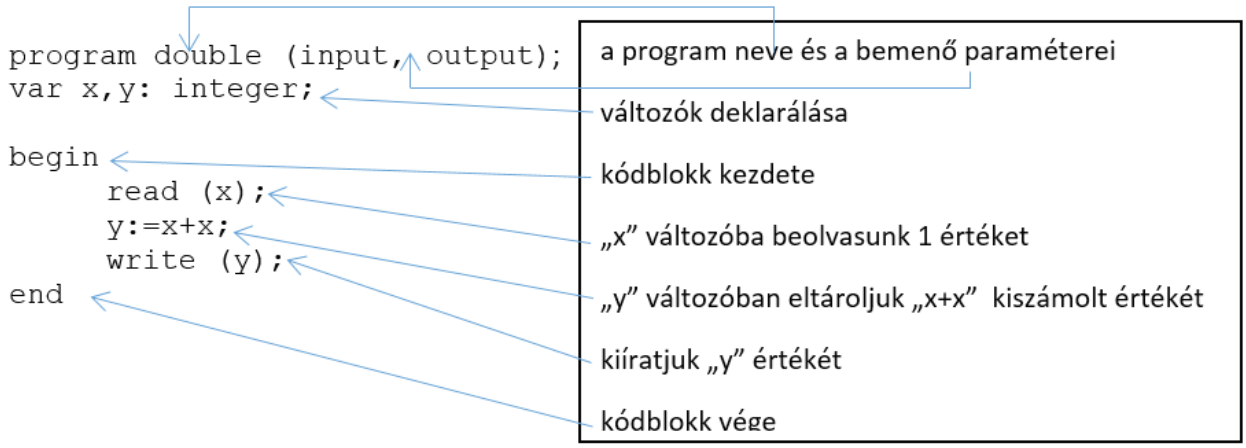
```
main( ) {
printf( "I want to hug my computer!" );
}
```

Látható, hogy a „main()" itt is megjelent, sőt a „{}"kapcsos zárójelpár is. A „main" egy függvény, amit a mögötte lévő „()" gömbölyű zárójelpár mutat. A neve annyiból különleges, hogy egy ilyen nevű függvénynek lennie kell a programunkban, ebben kezdődik a kód. Ez több modulnál érdekes. A „()" gömbölyű zárójelekben lehet felsorolni a függvény paramétereit, ha vannak. A „{}" kapcsos zárójelben pedig a kódunkat kell írni. Jelen esetben ez egy utasítás, a printf(). A printf is függvény, mögötte állnak a „()" gömbölyű zárójelek. De ennek a függvénynek van paramétere, méghozzá az, amit ki akarunk íratni. Mind a „main", mind a „printf" gyári, beépített függvények. Készíthetünk saját függvényeket, és használhatjuk a nyelv által rendelkezésünkre bocsátottakat is.

Bizonyos nyelvek nem a "{}" kapcsos zárójelpárt használják, hanem "begin" és "end" kulcsszavakkal jelzik a kódblokk kezdetét és végét. Ilyenkor általában a "main" is hiányzik.

Példa 2

A következő példa egy bekért szám kétszeresét írja ki:



Példa 3

A B.3.5.2 fejezetben használt szelekció a következő volt:

```
if (eredmeny >= 45)
        printf("Sikerult");
else
        printf("Nem sikerult");
```

Ez a részlet önmagában annyit csinál, hogy megvizsgálja az eredmeny nevű változó értékét, ha az nagyobb egyenlő 45, akkor kiírja, hogy „Sikerult", egyébként a „Nem sikerult" szöveg fog megjelenni.

Ez a kódrészlet egészében az alábbi ábrán látható. A változó értékadására egy beolvasást használunk, mint a 2. példában, illetve kommentezve megjelenik a kódban egy közvetlen értékadás. Természetesen csak az egyiknek van értelme, a kommentet jelentő „//" kettő perjel kitörölhető vagy bármelyik sor elé rakható.

```
main(){
        var eredmeny: Integer;
        // eredmeny = 21;
        read(eredmeny);
        if (eredmeny >= 45)
        printf(“Sikerult”);
else
                printf(“Nem sikerult”);
}
```

3.8.2. Kódhibák, és javításuk

A programozási hibák három fő csoportba oszthatók

1. Fordítási hibák
2. Futásidejű hibák
3. Logikai hibák

Fordítási hibák

A fordítási hibák olyan hibák, amelyek megakadályozzák a program futtatását. A legtöbb fordítási hiba a kód begépelése során keletkezik. Például egy kulcsszó elírása, néhány szükséges írásjel kihagyása, vagy az **Else** utasítás, oly módon történő használata, hogy előtte nem használt **If** parancsot. Elég csak egy „;” pontosvessző jelet lehagyni, máris nem fog lefordulni a program.

A következő kódrészlet nem működik megfelelően, mert hiányzik egy kulcsfontosságú írásjel;

```
if (result >= 45)
        printf(“Pass”)
else
        printf(“Fail”);
```

A helyes kódnak így kell kinéznie:

Hiányzott a pontos vessző, ami az utasítást zárja le. Azonban nem szabad mindenhova rakni. Az if és az else sorok mögött akkor lenne hiba, ha kiraknánk!

```
if (result >= 45)
        printf(“Pass”);
else
        printf(“Fail”);
```

Futásidejű hibák

A futásidejű hibák akkor jönnek elő, amikor fut a program. Ezek általában akkor jelentkeznek, amikor a program megpróbál végrehajtani egy olyan műveletet, amit nem lehet elvégezni.

Egy példa erre a nullával való osztás. Nézzük meg a következő példát:

Sebesség = Megtett út / Idő

Ha az **Idő** változó értéke 0, az osztás művelet sikertelen, és ez egy futásidejű hibát okoz. A fiatalabb nyelvek (pl.: C#, Java) ilyenkor kivételt dobnak. Ezeket fel kell ismerni, és kezelni kell. A programnak futnia kell ahhoz, hogy ezt a hibát észleljük, és ha az **Idő** érvényes értéket tartalmaz, például 1 vagy 2, akkor ez a hiba egyáltalán nem történik meg.

Logikai hibák

A logikai hibák olyan hibák, amelyek megakadályozzák, hogy a program azt csinálja, amire tervezték. A fordítási hibákat nevezzük szintaktikai hibának, ezeket a logikai hibákat pedig szemantikai hibáknak. A kódot össze lehet állítani, és hiba nélkül le is futtatható, de a műveletnek az eredménye nem az lesz, amit vártunk.

Az alábbi kód a korábban bemutatott verzió, de itt hibás. A bekért változó dupláját írja ki a program eredetileg:

```
1 program double (input, output);
2 var x,y: integer;
3 begin
4 read (x);
5 y:=x*x;            (// * szorzás az összeadás helyett +)
6 write (y);
7 end
```

A hiba a programkód 5. sorában található, * operátort használt + helyett.

Ez a kis kód jól mutatja, hogy hibás tesztadat választása esetén nem jövünk rá bizonyos hibákra. Ha 2 a bemenet, azt hihetnénk, hogy jól működik a program. Több adattal észrevehető a hiba. A program szintaktikailag hibátlan, lefordul, fut, számol, csak éppen nem azt, amit mi szerettünk volna.

3.8.1. Tesztkérdések

1.) Melyik állítás a helyes?
   i. Az osztály az objektumból jön létre.
   ii. Az objektum az osztályból jön létre
   iii. Az osztály a példányból jön létre.
   iv. A példány az objektumból jön létre.

2.) Melyik állítás a helyes?
   i. Az ősosztályok metódusai az öröklés során felülírhatóak az utódban.
   ii. Az utódosztályok metódusai az öröklés során felülírhatóak az ősben.
   iii. Az ősosztályok és az utódosztályok metódusai az öröklés során nem írhatóak felül.

3.) Melyik utasítást NEM használhatjuk strukturált programozás során?
   i. break
   ii. goto
   iii. return
   iv. continue

4.) Melyik állítások igazak az alábbiak közül?
   i. A tömb nem tartalmazhat objektumot
   ii. A tömb tartalmazhat objektumot
   iii. Egy tömbön belül bármilyen típusokat használhatunk
   iv. Egy tömbön belül azonos típusokat használhatunk

5.) Melyik NEM algoritmus leíró eszköz?
   i. folyamatábra
   ii. struktogram
   iii. UML
   iv. pszeudó-kód

6.) Melyik állítások igazak?
   i. A függvény és az eljárás között az a különbség, hogy az eljárásnak nem lehet paramétere
   ii. Egy objektum metódusai függvények és eljárások is lehetnek
   iii. Egy függvénynek vagy eljárásnak átadott paraméter nem lehet ugyan olyan nevű és típusú, mint amit már korábban máshol is deklaráltunk
   iv. A függvény és az eljárás között az a különbség, hogy az eljárásnak nem lehet visszatérési értéke

7.) Melyik állítás igaz?
   i. Egy referencia típus vagy referenciaként átadott paraméter, hatással van arra a memóriaterületre, amire referenciát ad
   ii. Egy referencia típus vagy referenciaként átadott paraméter, nincs hatással arra a memóriaterületre, amire referenciát ad
   iii. Az egyszerű vagy más néven primitív típusok nem lehetnek referenciák

8.) Melyik NEM tesztelési módszer?
   i. feketedoboz
   ii. fehérdoboz
   iii. átlátszódoboz

9.) Melyik kódsor hibás?
   i. printf(„hiba”);
   ii. for(i=1,i<5,i++)
   iii. if(valami < 13)
   iv. if(valami < 13) printf(„kisebb”); else printf(„nagyobb”);

10.) Melyik kódsor az, ami lefordul, de logikai hibát tartalmaz?
   i. if(valami < 13); printf(„kisebb”);
   ii. printf(„hiba”);
   iii. for(i=1;i<5;i++)
   iv. if(valami < 13) printf(„kisebb”);

3.8.2. Megoldások

1. ii.
2. i.
3. ii.
4. ii. és iv.
5. iii.
6. ii. és iv.
7. i.
8. iii.
9. ii.
10. i.

4. Felhasználói interfész és webtervezés

4.1. Ember-gép interakció: irányelvek és szabványok

Tanulási célok

- Meghatározza a kommunikációelmélet alapfogalmait, mint a küldő, az üzenet és a fogadó.
- Érti, hogy hogyan épül be a kommunikáció az emberek életébe, és felismeri a kommunikáció (információközlés) hatékony módjait.
- Meghatározza a felhasználói interfész fogalmát, és felsorolja a különböző típusait, mint a szöveges, a grafikus és a hangalapú.
- Vázolja az emberi érzékszervek felé való információközlésre alkalmas technológiákat, mint a hangtípusok, a vizuális jelek, a digitális szagok és az érintés/tapintás.
- Felsorol modelleket, amelyekkel egy felhasználói interfész hatékonysága tesztelhető, annak kialakítási követelményei és céljai tükrében.

4.1.1. A kommunikációelmélet alapfogalmai

A kommunikáció leírható, mint adatátviteli folyamat. A kommunikációelmélet ennek a folyamatnak a működését vizsgálja.

A kommunikáció résztvevői a *küldő*, az *üzenet* és a *vevő*.

A küldő az eredeti üzenet *információforrása*, aki kódolja, majd továbbítja az üzenetet a vevőhöz.

Az üzenet, az az *adat*, amit az információforrás küld a vevőnek, azaz a *tartalom*.

A vevő az üzenet címzettje, ez a *közönsége* az üzenetnek.

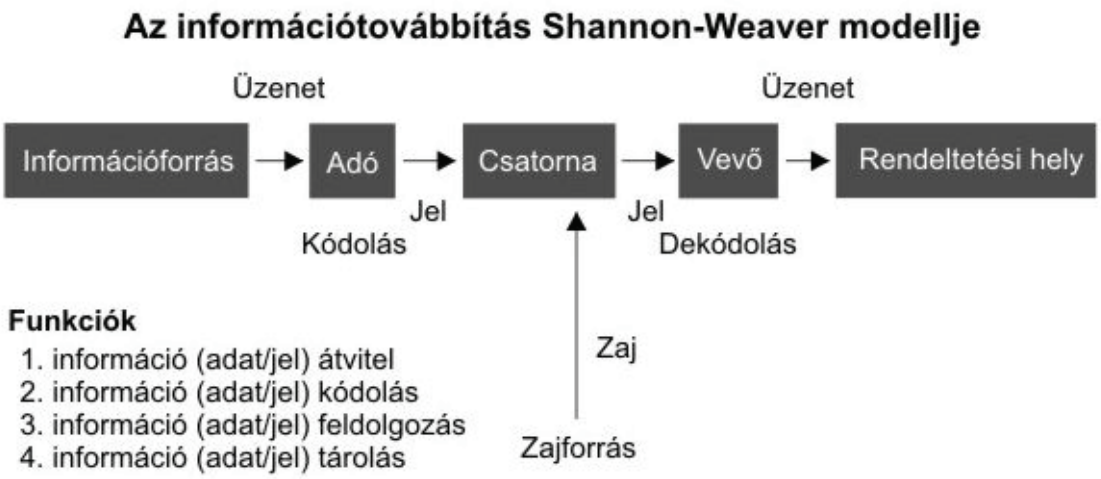
Értelmezésnek nevezünk minden olyan műveletet, melyet a vevő végez az üzenet dekódolása és megértése érdekében.

A médium (közeg) az üzenet továbbításának a közvetítő eleme.

Zajnak nevezünk minden olyan dolgot, ami az üzenet tisztaságát zavarja, és ami torzítja az üzenetet a küldő és a vevő között.

Az 1948-ban, Shannon által leírt kommunikációs folyamat az alábbi 8 részből áll:

- Az információforrás az *a személy, aki* létrehozza az üzenetet
- Az üzenet az, amit az információ forrása a vevőnek elküld.
- Az adó két ember közötti kommunikáció esetén ez a száj, mely a hangokat, és a test, ami a gesztusokat továbbítja.
- A jel az, ami a csatornán áthalad.
- hordozó, vagy más néven csatorna, lehet a levegő, a fény, az elektromosság, papír vagy valamilyen üzenetküldő rendszer.
- A zajok olyan másodlagos jelek, melyek zavarják vagy torzítják az elküldött jelsort.
- Vevő. A *szemtől szembe* zajló kommunikáció során a fül kapja a hangokat és a szem a gesztusokat.
- célállomás: Feltehetőleg egy személy, aki megkapja és feldolgozza az üzenetet.



4.1 ábra Kommunikációs modell
(forrás: Komenczi Bertalan: Információelmélet (2011))

4.1.2. Kommunikáció hatékony módjai

A kommunikáció leghatékonyabb eszköze a *face to face* kapcsolat. Ebben részt vesz az összes érzékünk, a látás, a hallás és a szaglás. Ehhez további lényeges elem is társul: a testbeszéd és arcmimika, ami által jól láthatóvá válik, hogy mit próbálunk átadni.

Az elektronikus kommunikáció az a módszer, mellyel a különféle információt elektromos energia formájában, a fény sebességével közölnek. Az eredeti információt (hang, fény, mechanikai energia…) tehát előbb mindig át kell alakítani elektromos energiává, amit azután vezetékeken, kábelen továbbítanak, vagy elektromágneses hullámok formájában a térbe sugároznak (nem minden szakaszon halad feltétlenül elektronikusan, mehet fény formájában, üvegszálban stb.). A vevő az elektromos energiát visszaalakítja eredeti (vagy más, az ember vagy a feldolgozó gép által értelmezhető) formájába.

Típusai

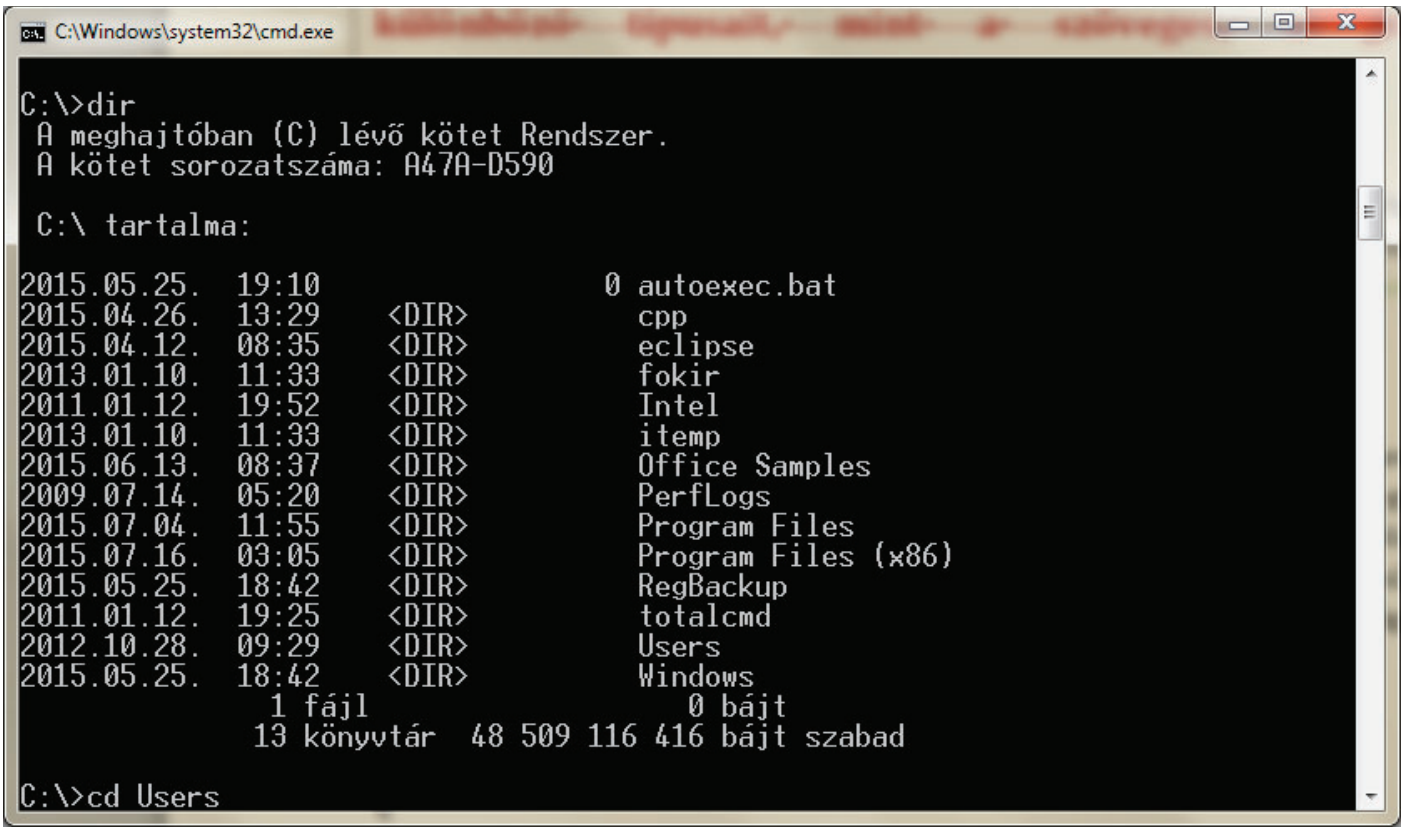
- elektronikus levél (e-mail)
- levelezőlista (mailing list)
- vitacsoport (discussion group)
- csevegő programok (chat)
- telefonálás az interneten keresztül (netphone)
- távoli bejelentkezés (telnet) A saját számítógépéről be tud jelentkezni egy másik (mindegy, hogy a világ melyik részén lévő) számítógépre. A klasszikus felhasználás, hogy ha például külföldön van, bármelyik gépről be tud jelentkezni az Internet szolgáltatója gépére, és el tudja olvasni a leveleit, új leveleket tud írni stb., anélkül, hogy nemzetközi telefonhívást végezne.
- webkamera
- videokonferencia

4.1.3. Felhasználói interfész fogalma

A felhasználói felület a számítógép azon része, ahol egy szoftveren keresztül a felhasználó láthat, érinthet, hallhat vagy beszélhet. Ezeknek az összessége teszi lehetővé, hogy a felhasználó és a számítógép kommunikáljon egymással. A felhasználói felület az az eszköz, amin keresztül kommunikálni tudunk a számítógéppel és a gép is velünk.

Karakteres felület

A kezdeti időkben a számítógép felhasználók soronként gépelt szöveget (parancsokat) használtak, ami zöld vagy fehér betűvel fekete háttér mellett látszódott. Tudni kellett, hogy a gép hogyan dolgozik, milyen parancsokat ismer, és hogyan kell ezeket a parancsokat pontosan írni. Ezt a felületet csak képzett számítógépes szakemberek tudták használni.

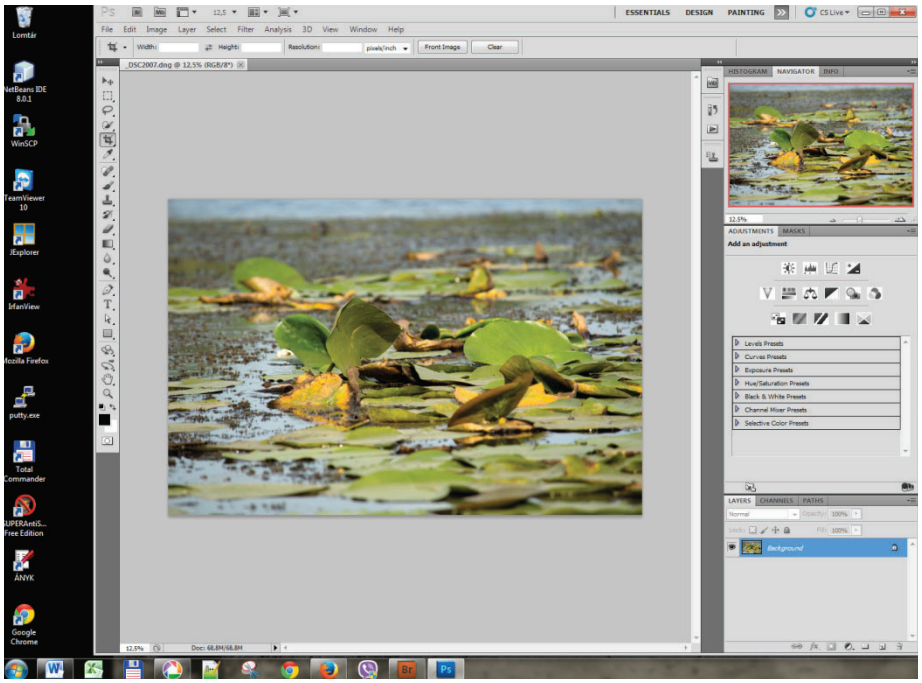


4.1. ábra Karakteres felhasználói felület

Azonban felmerült az igény olyan felület kialakítására, ami mindenki számára könnyen használható és érthető. Ezután olyan programokat készítettetek, melyeken menük segítségével felhasználóbarátabbá vált a kommunikációs felület. Minden alkalmazásnak megvolt a maga sajátos felülete, és kevés hasonlóság volt az alkalmazások között. A számítógép még mindig csak azoknak a magasan képzett szakembereknek és a lelkes érdeklődőknek volt hatékonyan használható, akik hajlandóak voltak jelentős időt fektetni a tanulásra, és alkalmazkodni tudtak a számítógép különleges igényeihez és viselkedéséhez.

Grafikus felhasználó felület

A modern grafikus felhasználói felület, azaz a GUI (Graphics User Interface) fejlődésének köszönhetően egyre könnyebben elérhetővé vált a számítógépes technológia a vállalkozások és a magánszemélyek számára. Az egyszerűbb számítógép használat érdekében a grafikus felhasználói felület ikonokat alkalmaz és egér használatot is biztosít. A GUI három fő összetevője az egérkurzor, az ikonok, és az azonos felépítésű ablakok és menürendszer.



4.2. ábra Grafikus felhasználói felület

Hangalapú felhasználói felületek

Ilyen felhasználói felület esetén a számítógép bemenetként hangutasításokat fogad, a kimeneti oldalon pedig az utasításoknak megfelelő viselkedést biztosítja. A felhasználó a bevitelt gombok megnyomásával vagy verbálisan végzi az adott felületen.

Info

Manapság már léteznek olyan interfészek és szoftverek, melyek biztosítják a hangalapú vezérlés lehetőségét, valamint a számítógépek is sok mindenre képesek a felhasználói parancsoknak megfelelően az egyszerű hangok kibocsájtástól a szövegek felolvasásáig, ami például esélyegyenlőséget biztosíthat látáskorlátozottak számára.

4.1.4. Ahogy a gépek információt közölhetnek az emberrel

Általánosságban a számítógépek a monitoron keresztül jelenítik meg számunkra az információt, de lehetőségünk van kinyomtatni is a látottakat. További lehetőség a hangszórók használata a hangok „megjelenítéséhez".

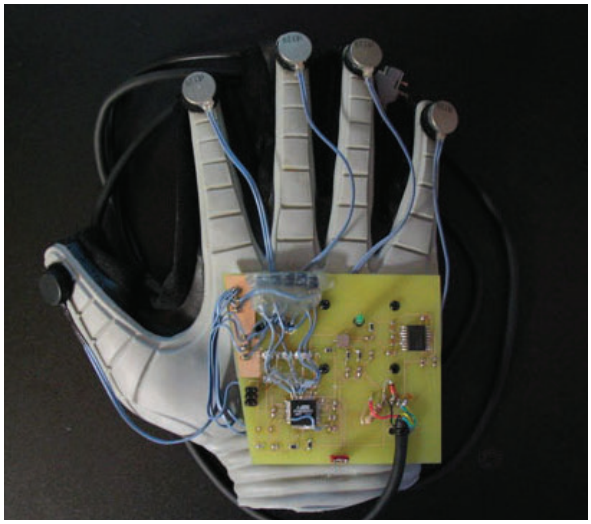
Manapság azonban több technikát is kifejlesztettek acélból, hogy a számítógép kommunikálni tudjon velünk.

Vizuális jeleket használnak például a kommunikációban olyan alkalmazásoknál, amikor a hangjelzés problémát jelenthet az olvasónak. Ilyenkor a vizuális jelek jelennek meg a képernyőn, ha az egeret egy ikon, vagy egy alkalmazás szövege fölé visszük, vagy ha szörfözünk egy weboldalon.

Jelenleg kidolgozás alatt van az a technológia, amely a digitális illatokkal kísérletezik. Ez lehetővé teszi, hogy a számítógép felhasználó a gépen keresztül érezze az illatokat és szagokat. A vállalatok alig várják, hogy a végre kipróbálhassák ezt az új értékesítési technikát. Az ötlet az, hogy illata lehessen filmeknek, játékoknak, zenéknek, animációknak, vagy bármilyen digitális médiának lehetőséget ad arra, hogy egy új ingert juttasson el a nézőknek, a játékosoknak, és a zenerajongóknak. A hangulatot, érzelmeket és a karaktereket mind fokozni lehet az illatok által. Az elképzelés szerint létrehozzák a digitális szagok adatbázisát, és továbbítják azokat a felhasználónak számítógépén vagy házimozi rendszerén keresztül.

A Haptic feedback az érintés (tapintás) érzékelésének, ellenőrzésének és a számítógépes alkalmazásokkal való kölcsönhatásának tudománya. A speciális bemeneti / kimeneti eszközökkel, mint például a joystick, adat kesztyű vagy más eszközök, a felhasználók alkalmazásában rezgésérzet formájában visszajelzést kapnak

arról, hogy mit érezhet a kéz, vagy más testrész. A vizuális megjelenítést és a tapintás-technológiát egymással kombinálva olyan feladatokra használható fel, amelyek szem-kéz koordinációt igényelnek, mint például a műtét és űrhajós manőverek, továbbá játékokban is, ahol érezhetőek és láthatóak is az interakciók. Érdekes példa lehet erre, egy teniszmeccs egy olyan számítógép felhasználóval, aki a világ valamely másik részén van. Mindketten láthatják a mozgó labdát, és a tapintás biztosító eszköz segítségével érzékelhetik a teniszütő pozícióját, lendületét, és a labdára kifejtett hatását.



4.3. ábra Haptic feedback

4.1.5. Felhasználói felület hatékonysága

A használhatóság egy olyan minőségi mutató, amely azt jellemzi, hogy a felhasználói felületeket mennyire egyszerűen lehet használni.

Használhatósági paraméterek:

- Megtanulhatóság (Learnability): Amikor a felhasználó először találkozik a felülettel, mennyire egyszerű számára az alapvető feladatok elvégzése?
- Hatékonyság (Efficiency): Ha a felhasználó már megismerkedett a felülettel, akkor milyen gyorsan tud különböző feladatokat megoldani?
- Megjegyezhetőség (Memorability): Amennyiben a felhasználó egy ideig nem használta a felületet, a korábban megszerzett tudását mennyire gyorsan tudja újra felidézni?
- Hibák (Errors): Hány hibát vét a felhasználó, milyen mértékűek ezek, és mennyire tudják a hibáikat könnyen javítani?
- Elégedettség (Satisfaction): Mennyire megfelelő, kényelmes a felület használata a felhasználó számára?

Tesztelési módszerek:

A **papír prototípus tesztelésben** nem valódi működő oldalt vagy alkalmazást tesztelünk, hanem rajzolt vagy nyomtatott képekkel imitáljuk annak működését.

A **drótváz tervek** megmutatják, hogy a felületen milyen összetevők milyen sorrendben és pozíción helyezkednek el. Ez ergonómiai szempontból azért is előnyös, mert már a drótváz-tervekből is kiderül, ha a felület túl bonyolult, átláthatatlan, ezért még a konkrét grafikai tervek megvalósítása előtt lehetőség van a módosításra.

A felület tesztelésbe már a tervezés során vonjuk be a majdani felhasználókat is. A tesztelést a fejlesztés menetével összhangban többször is el kell végezni, és a teszteredmények alapján módosítani kell a felhasználói felületet.

Természetesen sort kell keríteni egy végső tesztre a prototípuson az átadás előtt.

4.2. Grafikai tervezés

Tanulási célok

- Felvázolja a grafika és az animáció (pixel- és vektorgrafika), a digitális hang és videó fogalmait, valamint bemutatja a köztük rejlő különbségeket, használatukat, illetve szabványos formátumaikat.
- Bemutatja a rajzok, a képek, a színek és az animációk használatának előnyeit, és alkalmazza a grafikai tervezés alapelveit, mint az egyensúly, a harmónia, a kontraszt és a változatosság.
- Használja az egyszerű képmanipulációkra elterjedt eszközöket a méret, az alak, a színek, a kontraszt és az átlátszóság terén.

4.2.1. Multimédiás elemek jellemzői

Képformátumok

A képek tárolásuk és készítésük szempontjából két típusba sorolhatók: lehetnek pixelgrafikus és vektorgrafikus képek.

Pixelgrafikus (Bittérképes) képek rácshálóban elhelyezett képpontokból (pixelekből) állnak. A kép mindegyik pixele a megjelenítés színével kapcsolatos információt tartalmaz. A pixelgrafikus képek jellemzője a felbontás (ppi – plot per inch), mely meghatározza a kép inchenkénti képpontjainak számát. Mindezek következményeként a kép méretének növelése a képminőség romlását eredményezi.

4.4. ábra Pixelgrafikus ábra

A pixelgrafikus képformátumok három fő kategóriába sorolhatóak:

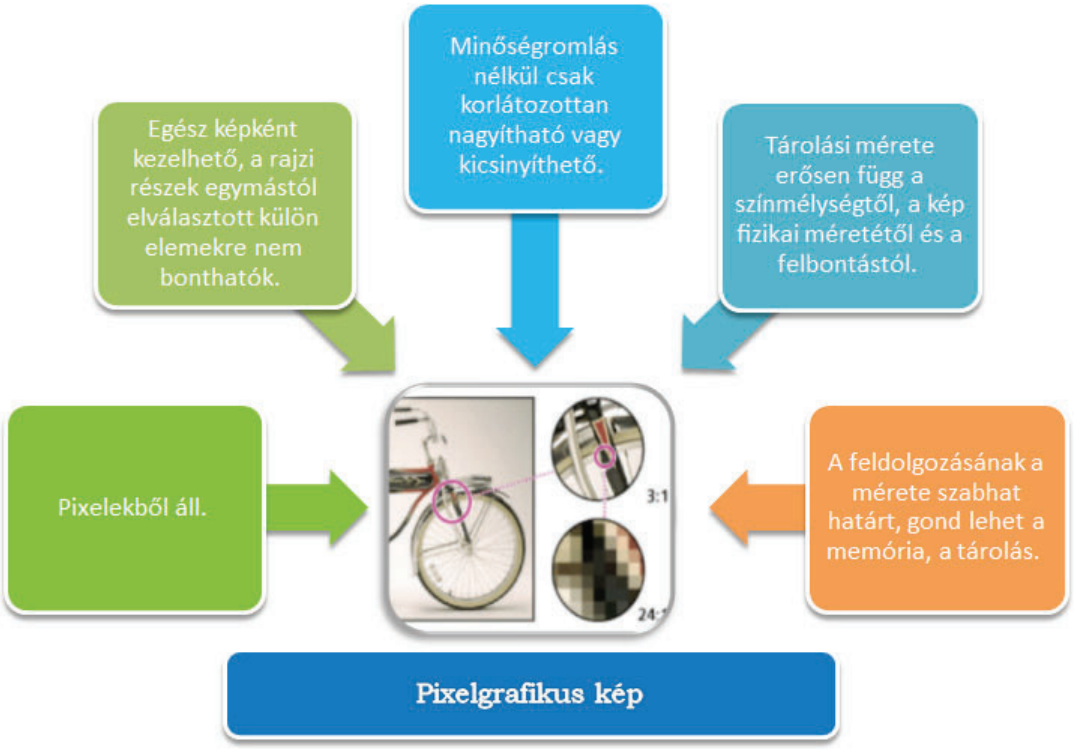
- Tömörítés nélküli formátumok, mint pl. a BMP, PICT, TIFF;
- Veszteségmentes tömörítés formátumok: GIF, PNG;
- Veszteséges tömörítés formátumok: JPG.

A legtöbb pixelgrafikus képet könnyedén átalakíthatjuk egy másik pixelgrafikus formátumba. A pixelgrafikus képek mérete a képpontok számától, illetve a képen használt színek számától függ. Méretük igen nagy is lehet, ezért gyakran tömörítik, hogy a méretüket redukálják. A tömörítést végezhetjük információ-veszteség nélkül, ám képek (hangok és filmek) esetében lehetőség van a veszteséges tömörítésre is. Ez talán első hallásra meglepőnek tűnhet. A tömörítési eljárás azon alapul, hogy érzékszerveink tökéletlenek, és

nem vesszük észre a képek apró minőségromlását.

Általában az interneten pixelgrafikus ábrák tárolására a PNG, vagy (kevés szín alkalmazása esetén) a GIF formátum a használatos, fotók esetében a JPG formátum a használatos, festményekről készült fényképek esetén a JPG, PNG formátumokat használják.

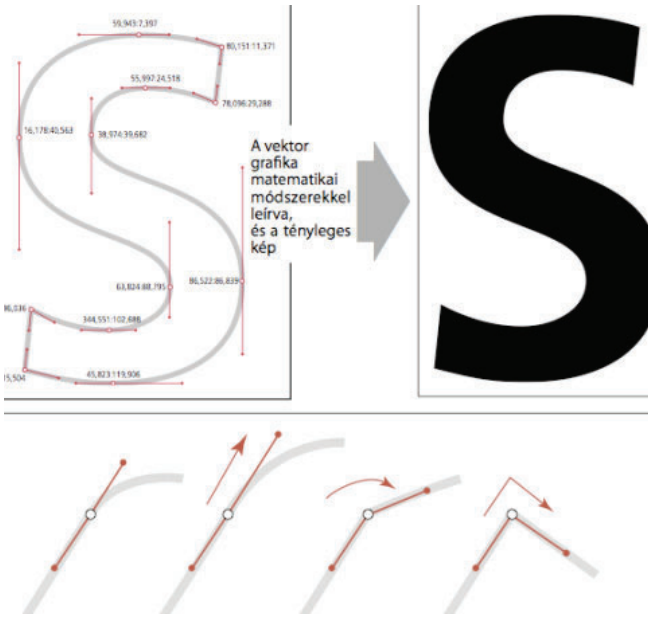
A GIF formátum alkalmas arra, hogy több képet is tároljon egyetlen fájlban, majd ezeket egymás után megjelenítse az arra alkalmas megjelenítő program (pl. a böngésző). Az animált GIF ezáltal alkalmas rövid animációk, folyamatok bemutatására is.



4.5. ábra Pixelgrafikus kép jellemzői

Vektorgrafika

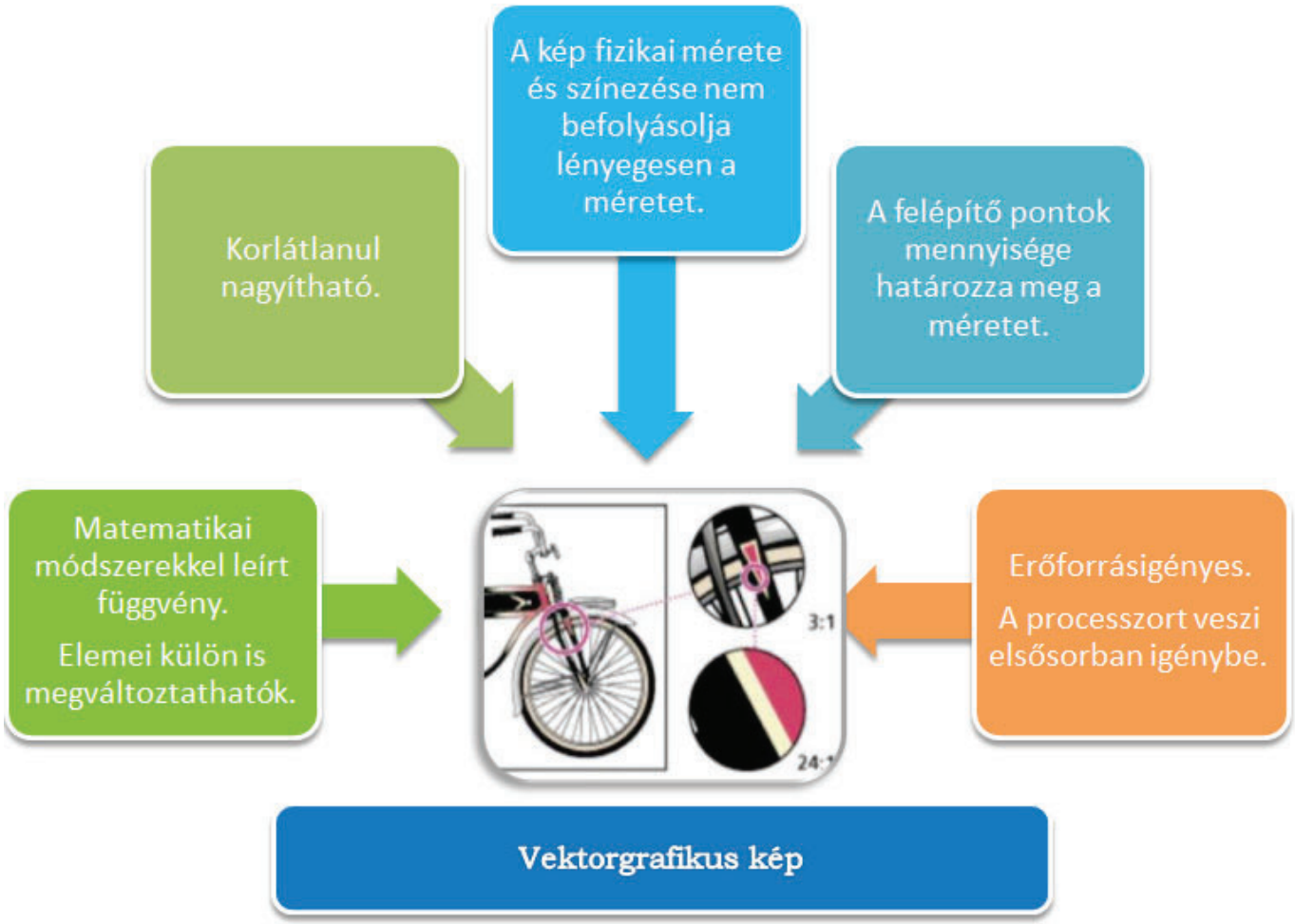
Vektorgrafika sok egyedi objektumból épül fel. Ezeket az objektumokat valamilyen matematikai képlettel, formulával határozzák meg, és egyedi tulajdonságokat rendelnek hozzá, mint például a szín, a kitöltés és a körvonal.



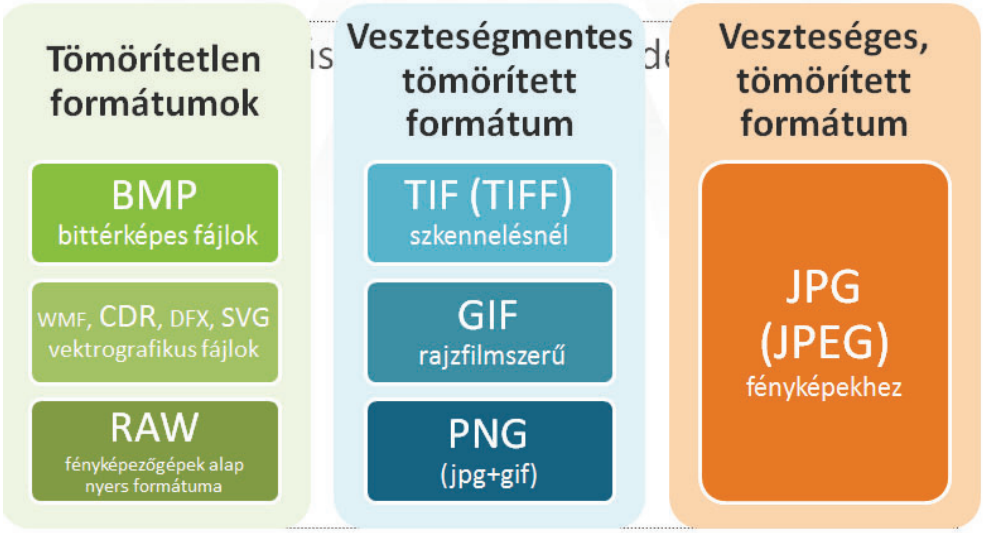
4.6. ábra Vektrografikus ábra

Vektorgrafika független a felbontástól, a kép tetszőlegesen méretben is a legjobb minőséget nyújtja.

Gyakori vektoros formátum például, az AI (Adobe Illustrator), CDR (CorelDraw), CGM (Computer Graphics Metafile), SWF (Shockwave Flash), és DXF (AutoCAD és más CAD szoftverek). Vektorgrafika esetében a fájlméret a felhasznált objektumok számától függ.



4.7. ábra Vektorgrafikus kép jellemzői



4.8. ábra Képformátumok – összefoglaló.

Hangformátumok

Digitális hangformátum esetén az analóg jelet digitális jellé kell alakítani, adott mintavételi arány és bit felbontás mellett. Általánosságban elmondható, hogy a magasabb mintavételezési arány és bit felbontás a nagyobb eredethűséget eredményez, valamint megnöveli a digitális adatmennyiséget, azaz növeli a fájlméretet.

A digitális felvétel során használt eszköz egy analóg-digitális átalakító (ADC). Az ADC rögzíti a hangsorozaton az elektromos feszültség egy pillanatfelvételét, és egy digitális számként küldi tovább a számítógépnek.

Info **Ha feszültséget másodpercenként ezerszer rögzítjük, akkor nagyon jól megközelíthetjük az eredeti hangjelet.**

A hangformátumok három fő kategóriába sorolhatóak:
- Tömörítés nélküli formátumok;
- Veszteségmentes tömörítés formátumok;
- Veszteséges tömörítés formátumok.

A tömörítés nélküli hangformátumok (PCM) ahogy a neve is mutatja – egy olyan formátum, ami nem használ a hang tárolása során semmilyen tömörítési eljárást, minden egyes adatot egymástól függetlenül tárolnak el. Ennek következménye a nagyobb fájlméret. Ilyen például a WAV hangfájl.

A veszteségmentes tömörítés esetében a tömörítés során nem történik információvesztés, ebből az eredeti tömörítetlen változat visszaállítható, és a digitális hangállomány minősége nem romlik. A WMA hangállomány-formátum veszteségmentes tömörítést használ.

A veszteséges tömörítés bizonyos mértékű adatvesztést eredményez, amikor a tömörítési algoritmus a szükségtelen információt kiküszöböli. Ez az eljárás a hang minőségromlását eredményezi, amit kisebb-nagyobb mértékben érzékelhetünk is, amikor meghallgatjuk a felvételt. Az így tömörített állományból nem állítható vissza az eredeti fájl. A veszteséges tömörítés a kis fájlméret miatt vált népszerűvé az online használat során, mert könnyen továbbítható az interneten. Az MP3 és a Real Audio veszteséges tömörítést használ.

A leggyakoribb Windows hangfájl formátumok a következőek:
- Windows Media Audio: WMA, WAV
- Real Audio: ra, ram, rm

A digitális videó egy olyan videó rögzítési rendszer, amely analóg helyett, digitális videójel felhasználásával működik. Ilyen formátumok például a MPEG-1 és MPEG-2.

4.2.2. Grafikai tervezés alapelvei

Az ügyfelek egy cégről az első benyomásukat a cég képi megjelenése alapján alkotják. Ezzel az arculattal találkozhatnak szórólapokon, Facebook-oldalon, a weben, online vagy nyomtatott hirdetésekben, névkártyákon, bélyegzőn. Fontos tehát, hogy milyen grafikai elemeket használunk az online vagy offline kommunikáció során.

Ezeket az elemeket foglalja rendszerbe és szabályozza az *arculati kézikönyv*. Itt definiáljuk a logót, a névjegykártya, a levélpapír és a boríték, az elektronikus hírlevél stb. felépítését, megadjuk a méreteket, elrendezéseket, a felhasznált betűtípusokat és színeket. A weblap tervezésekor mindig figyelembe kell venni az arculati kézikönyvben leírtakat.

Színek

A szín az életünk része:mindenhol színek vesznek körül bennünket, és hatnak ránk. Színek megfelelő használatával hatékonyan irányíthatjuk a látogatók figyelmét a honlapon. A jó színhasználat megkönnyítheti az olvasást, kellemessé teheti a böngészést az oldalon, ezáltal forgalomnövekedéshez vezet.

A grafikai tervezőnek tisztában kell lennie azzal, hogy melyik színt melyikkel együtt alkalmazhatja, vagy melyek azok, amelyek nemcsak, hogy nem illenek össze, de egyenesen olvashatatlanná is teszik a szöveget, ha a háttér az egyik, a betű a másik színben készül. Ilyen kombináció például a kék és piros. Általában **harmonikus kombinációkat** adnak azok a színek, amelyek a természetben is megtalálhatók egymás mellett.

Számtalan olyan alkalmazás található az Interneten, mely segíti az egymáshoz illő színek kiválasztását. Pl.: palettont.com

4.9. ábra Weblap színeinek tervezése - színpaletta

A komplementer színek a színkör két ellentétes oldalán helyezkednek el. A grafikai gyakorlatban háttérszín-betűszín kombinációban a komplementer színeket nem szabad együtt használni, mert látványuk vibráló hatást kelt, ezáltal olvashatatlanná válhat a szöveg, de semmiképpen sem bíztat olvasásra.

Komplementer színek

- Piros – Zöld
- Sárga – Ibolya
- Kék – Narancs

A színkörön szomszédos helyeket elfoglaló színek jól harmonizálnak egymással.

- Piros –Narancs
- Sárga – Zöld
- Kék – Ibolya

4.10. ábra Színek használata a weblapon

Általános elvként megfogalmazható, hogy a színekkel is óvatosan kell bánni. A színek meghatározzák az oldal alaphangulatát, és irányítják a figyelmet. Ne használjunk túl sokféle színt az oldalon, kerüljük a komplementer színek használatát, válasszunk inkább egymással jól harmonizáló színeket.

Képek használata a weboldalon

A képek a honlap fontos részei. Vizuális világban élünk, a grafikai elemek vonzzák a tekintetet. A képek egyedivé varázsolhatják az oldalt, meghatározzák a weblap hangulatát is. Azonban oda kell figyelnünk néhány dologra:

- Az első szempont, hogy a túl sok kép elvonja a figyelmet a fontos információról. A képek elsődleges szerepe a szöveges tartalom információtartalmának erősítése.
- A sok, felesleges, szükségtelenül nagyméretű kép lassíthatja az oldal letöltődését, ezáltal elveszthetjük látogatóinkat.
- Képeket használhatunk designelemként is, itt is fontos szempont a figyelem terelése, a lényeg kiemelése.

- Hátterek esetében olyan képet szabad csak használni, mely nem rontja a szöveg olvashatóságát.
- Az interneten használt képeket védi a szerzői jog. Olyan képeket használjunk weboldalainkon, melyek vagy saját készítésűek, vagy ingyenesen felhasználhatóak, vagy megvásároltuk őket!

4.2.3. Képek átalakításának leggyakoribb lehetőségei

Az internetes oldalakon használt multimédiás elemek nagyban meghatározzák az weblapok esztétikus megjelenését, valamint azok letöltési sebességét. A bájtokban kisebb kép, gyorsabban betöltődik. Érdemes megismerkedni tehát néhány grafikus program képkonverziós lehetőségeivel.

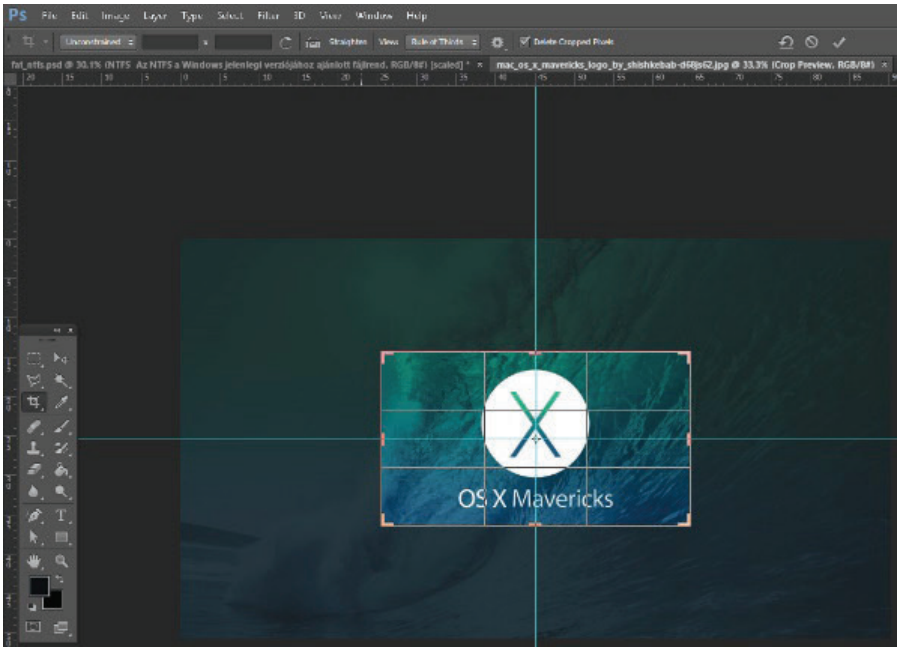
A képszerkesztés leghatékonyabb eszközei
- Adobe Photoshop
- Corel Paint
- Paint Shop Pro

Sok más különböző eszköz létezik még, kezdve az ingyenesektől, a nagyon drágákig.

A képek szerkesztésének leggyakoribb eszközei:

Vágás

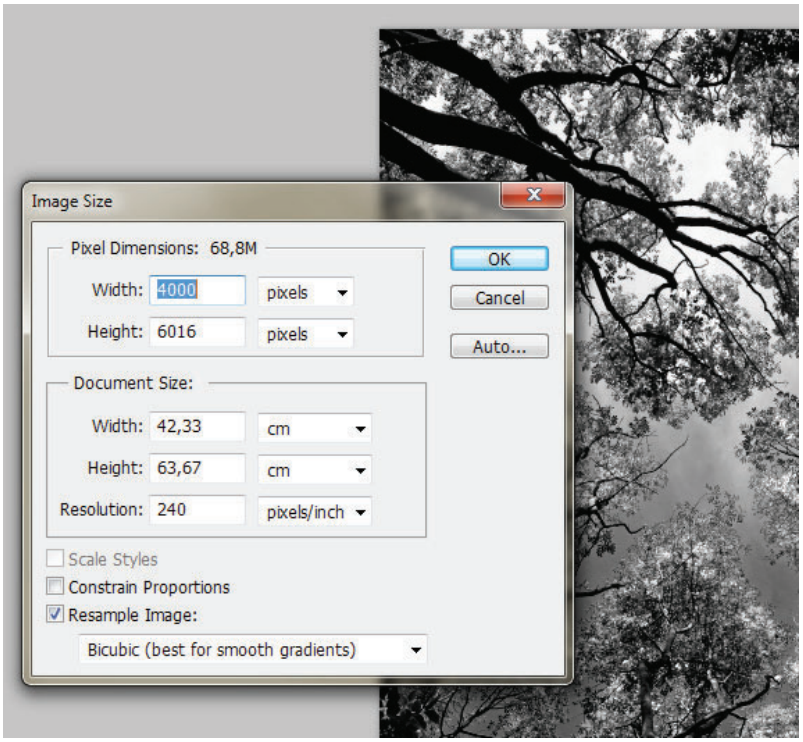
Képvágás: az az eljárás, amely során eltávolítják a kép széleinek egy részét, azért, hogy javítsák a kialakítást, hangsúlyozzák a kép tárgyát vagy megváltoztassák a kép méretarányát.



4.11. ábra Kép vágás

Átméretezés

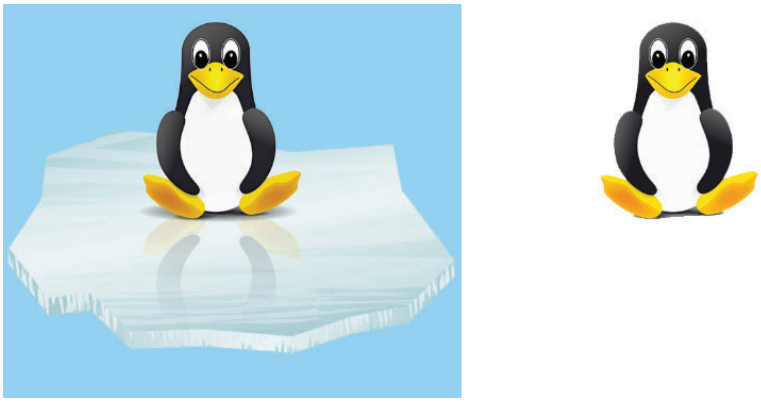
Kép méretezése során megváltoztatjuk a kép képpontjainak számát, és/vagy a bájtokban mért méretét a felhasználási célnak megfelelően.



4.12. ábra Kép átméretezése

Átlátszóság

A képek egyes részeinek átlátszóvá tételével a felhasználás során a kép mögötti háttér láthatóvá válik. (PNG formátum estén használható)

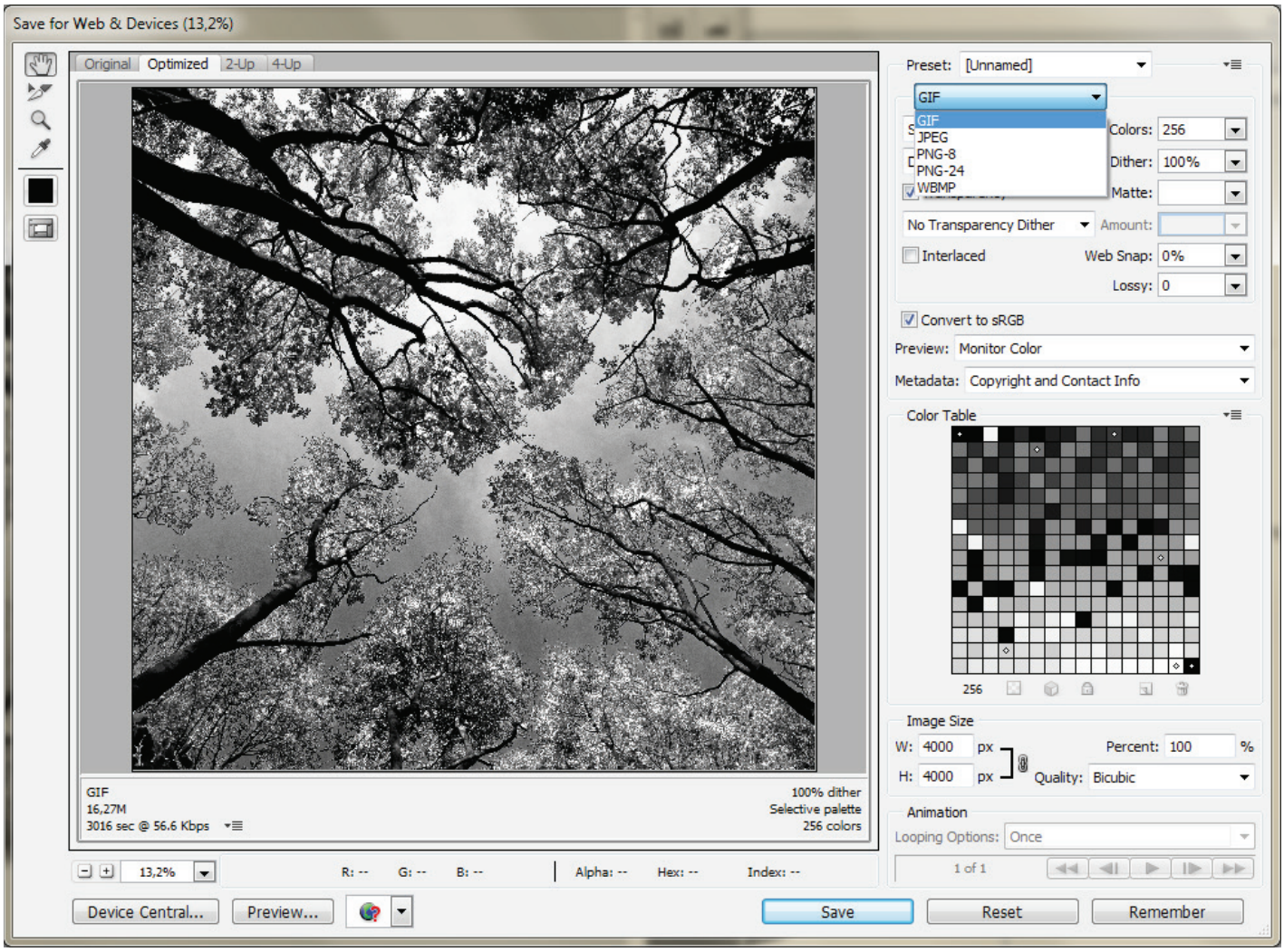


4.13. ábra Átlátszóság a képen

Átalakítás

A képeket egyik formátumból egy másikba konvertálhatjuk, pl. weblapon való felhasználáskor JPG, GIF, vagy PNG formátumba menthetjük, nyomdai használathoz a megfelelő színtérbe konvertálhatjuk stb.

Ezek a funkciók számos, külön erre a célra kifejlesztett ingyenes program segítségével is elérhetők: pl. Irfanview.



4.14. ábra Kép átalakítása webre

4.3. Web és hipermédia: lehetőségek és korlátok

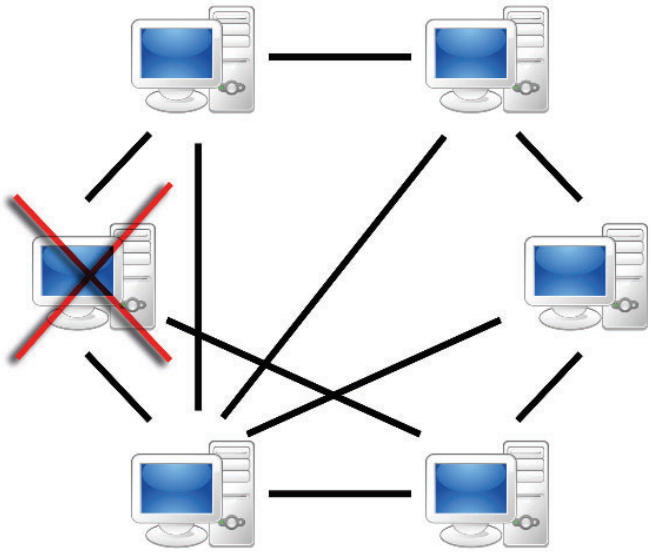
Tanulási célok

- Bemutatja az Internet és a WWW (World Wide Web) történetét.
- Meghatározza a hipertext és a hipermédia fogalmakat, és felvázolja fontosságukat a weboldalak tervezésében.
- Felvázolja a weboldalaknál használt gyakori összetevőket, mint a fejléc, az oldalsáv, az oldaltérkép, a kapcsolat, a keresési funkció, a segítség, az utolsó frissítés és a navigációs gombok.
- Bemutatja a belső és külső honlapok használatát és jelentőségét egy cégnél.
- Felvázolja egy üzleti honlap karbantartásának kihívásait.

4.3.1. Az Internet és a WWW (World Wide Web) története.

Az Internet használata előtt a legtöbb kommunikációs hálózat korlátozott volt. Az elterjedt számítógép hálózati módszer a központi nagyszámítógépes modellen alapult.

Az ARPANET-et, az Internetet ősét az amerikai hadsereg számára tervezték és építették ki. Tervezésekor fontos szempont volt, hogy egy olyan kommunikációs hálózatot hozzanak létre, ami még akkor is működne, ha az állomások közül néhányat megsemmisítene egy katonai támadás.

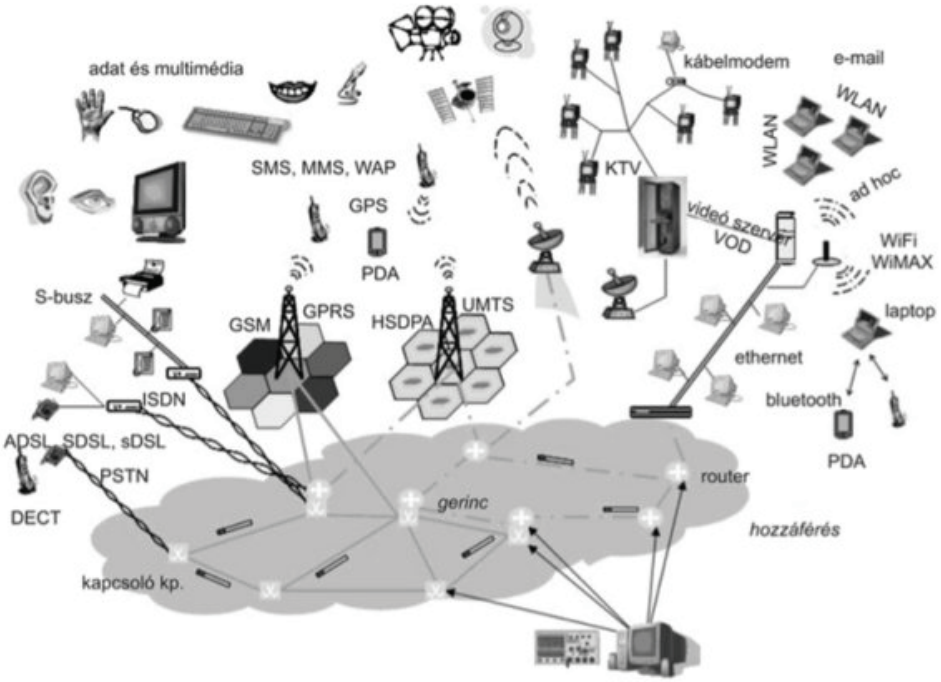


4.15. ábra Decentralizált hálózat

A korai Internetet számítógép szakértők, mérnökök, tudósok, és könyvtárosok használták Akkoriban nem voltak még otthoni vagy irodai számítógépek, és ha valaki használni akarta a netet, akkor annak egy nagyon bonyolult rendszert kellett meg tanulnia. Nem igazán volt felhasználóbarát.

A World Wide Web (WWW): az Internet egy szolgáltatása, mely a szerte a világban a weblapokon tárolt információk megjelenítésére szolgál.

A WWW és az Internet közötti különbség összemosódott. Az Internet azokból a számítógépekből és kommunikációs berendezésből áll, amiket arra használunk, hogy hozzáférhessünk a dokumentumokhoz a Weben.



A WWW-t először Tim Berners-Lee javasolta 1989-ben, miközben a CERN-nél, az Európai Nukleáris Kutatási Szervezetnél dolgozott Genfben. A CERN programkönyvtáron keresztül, egy korai WWW rendszert hoztak létre a részecskefizikus közösség számára.

Ez tartalmazott egy egyszerű böngészőt, internetes szerverszoftvert és egy számítógépes könyvtárat. Kezdetben az egyetemek és kutató-laboratóriumok kezdték el használni.

Nem sokkal később általánosan hozzáférhetővé vált az Interneten keresztül a WWW.

Az első internetes szerver az Egyesült Államokban lett online 1991 decemberében, a Stanford Linear Accelerator Centernél (SLAC) Kaliforniában, és 1993 végéig több mint 500 internetes szerverről tudunk.

4.16. ábra Internet története

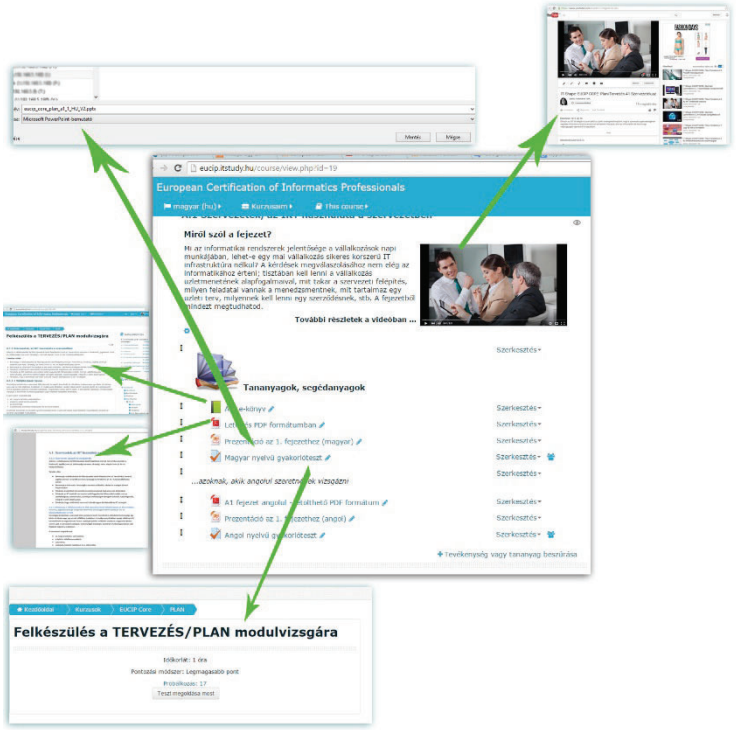
1995 januárjában, megalakult az International World Wide Web Consortium (W3C) amelynek az volt a feladata, hogy irányítsa a World Wide Web-et, kihasználja az összes benne rejlő lehetőséget. Olyan közös protokollok kidolgozását szorgalmazták, amelyek elősegítik a fejlődését, és biztosítják az Internet egyes szolgáltatásai között a kölcsönös átjárhatóságot.

4.17. ábra W3C konzorcium logója

4.3.2. Hipertext és hipermédia

A hipertext egy olyan módszer, melynek segítségével a számítógépes hálózaton elhelyezett információ nem lineáris formában, hanem ún. linkek, hivatkozások segítségével érhetők el. Más szóval a hipertext egy kapcsolattal rendelkező szöveg, egy másik szöveghez társítva.

Ahelyett, hogy az olvasás vagy a tanulás a szerző, szerkesztő vagy kiadó által egy előre definiált sorrendben történne, a hipertext olvasói a saját útjukat követhetik: saját bejárási sorrendeket hozhatnak létre és saját maguk értelmezhetik az anyagot.



4.18. ábra Beágyazott linkek az oldalon

Ez információs csomópontokba elhelyezett linkek létrehozásával érhető el. Ezek a linkek arra szolgálnak, hogy az olvasók a kérdéssel kapcsolatos további információkért oda ugorhassanak, ahol az adott témát bővebben tárgyalják. Ezen oldalon is lehet több link, amik minden egyes olvasót más-más irányba vihetnek tovább.

A hipermédia dokumentumok nemcsak egy további szövegrészhez tartalmazhatnak hivatkozásokat hanem, más típusú médiumokhoz - hangokhoz, képekhez, és filmekhez is. Maguk a képek is lehetnek hangokra vagy a dokumentumokra hivatkozó linkek. A hipermédia a hiperszöveg kiterjesztése, amely a szövegelemek mellett támogatja a kép-, hang- és videóelemeket is.

Fontos, hogy a felhasználók mindig tudják, hogy milyen úton navigáltak végig a weblapon. De az is elengedhetetlen, hogy ne kényszerítsük őket arra, hogy egy előre meghatározott útvonalat kövessenek.

Weboldal tervezésekor fontos szempont az oldal felhasználóbarát szerkezete és a jól átlátható navigáció lehetősége. Fontos, hogy a navigáció könnyen megvalósítható legyen, az olvasó ne vesszen el a weboldal böngészése során.

A fejlesztőnek szüksége van arra, hogy tiszta képet kapjon arról, hogy az emberek miért akarják meglátogatni az oldalt, és mit szándékoznak ott tenni.

Az alábbi kérdéseket kell figyelembe venni:

- Ha valaki egy konkrét információért keresi fel az oldalt, milyen egyszerűen tudja megtalálni azt?
- Honnan tudja a látogató, hogy mit láthat vagy csinálhat az oldalon?
- Honnan tudja egy látogató kitalálni, hogyan kell navigálni az adott oldalon?
- Honnan tudja a látogató, hogy látott-e már mindent az oldalon?
- Hogyan lehet a látogatóval közölni, hogy mi az, amit látott már az oldalon és mi az, amit még nem?

Fontos, hogy a látogató könnyen átláthassa, mik az újdonságok, és mik változtak meg a legutóbbi látogatásuk óta. Idővel néhány alapértelmezés elterjedt a webes navigáció kapcsán, és ezek olyan megszokottá váltak, hogy nem lenne bölcs dolog megváltoztatni őket.

Tipp **Például a színeket arra használhatjuk, hogy információt közvetítsünk vagy felhívjuk a figyelmet valami fontosra.**

4.3.3. Weboldalak alapvető egységei

4.19. ábra Weboldalak alapvető egységei

A weboldalnak több olyan része is van, amely minden weboldalnak szükséges eleme. Ilyen például a fejléc, a navigációs sávok, a kapcsolatfelvételi lehetőség, a súgó, keresés funkciók stb.

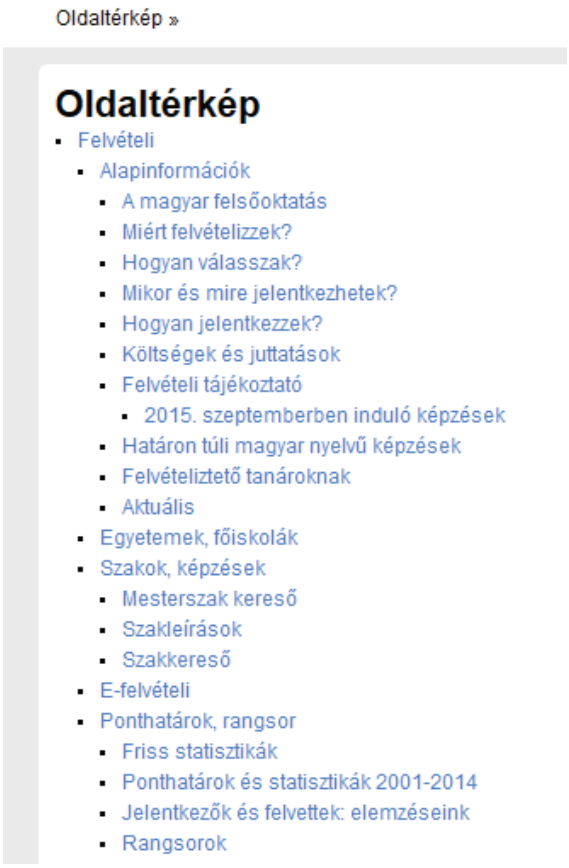
A **navigálásra** hagyományosan a navigációs sávokat használják. Ezek a sávok az oldalakon belül a megfelelő helyekre mutató linkekből állnak. Az egyes oldalak vízszintes sorokba vagy sávokba csoportosítják a hivatkozásokat.

4.20. ábra Navigációs sáv

Más oldalakon a linkek függőleges oszlopban helyezkednek el. Annak ellenére, hogy a navigációs sávok fontos részei az oldalnak, mivel a képernyőn sok helyet foglalnak, csökkentik az információra szánt helyet az oldalon. Ezért a navigációs sávnak a lehető legkisebb méretűnek kell lennie, úgy, hogy a feladatait továbbra is ellássa. Emiatt van, hogy a sávokban lévő szövegek kisebbek, mint az oldalon lévő szövegek.

A navigációs ikonokat arra használják, hogy a látogatókat a honlap egy másik részre irányítsák. Ezeknek az ikonoknak utalniuk kell arra az oldalra, amire mutatnak. A jó ikonok elsődleges célja, hogy anélkül sugallják a program vagy a művelet célját, hogy a felhasználó elolvasná a mellékelt szöveget. Az ikonok mellé meg kell adni egy alternatív szöveget, ami biztosítja a grafikus hivatkozások szöveges leírását.

Az **oldaltérkép** felsorolja az összes olyan oldal címét, ami a honlapon szerepel. A megfelelő linkre való kattintással a látogatók átirányíthatóak a honlap bármely oldalára. Az Oldaltérkép áttekintést nyújt az oldalak szerkezetéről és fő területeiről minden látogató számára.



4.21. ábra Oldaltérkép

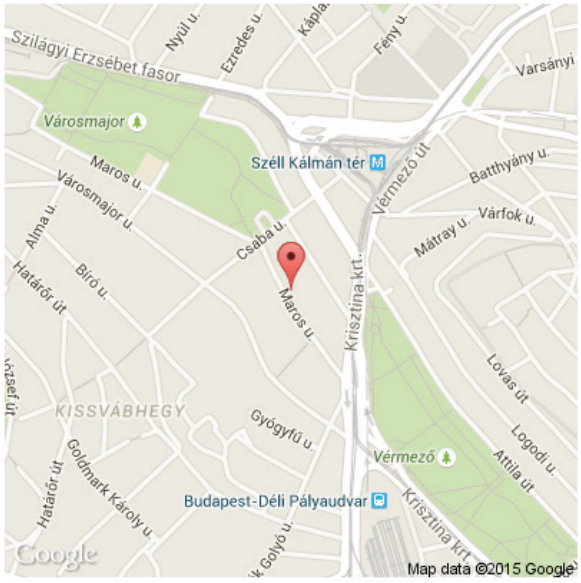
A **kapcsolati rész**, a weboldal tulajdonosának elérhetőségeit tartalmazza. Ide tartozik a telefonszám, az e-mail cím, és esetleg a postai cím (telephely, nyitva tartás, térkép).

Személyesen...

Cím: Budapest, XII. Maros utca 19-21. (Bejárat az utcáról.)

Nyitva tartás:
Hétfő, kedd, csütörtök, péntek: 8-16:30-ig
Szerda: 8-18-ig

Térkép:



4.22. ábra Kapcsolat

A kereső funkció lehetővé teszi a felhasználók számára a honlap különböző oldalai közötti keresés gyors és intuitív módját. Csakúgy, mint a keresőmotoroknál és az indexeléseknél, itt is a keresés alapja az egyes oldalakhoz tartozó címleírások és a kulcsszavak.

4.23. ábra Keresőmező

Egy jó weboldal tartalmaz **Súgó, vagy** „GYIK" (Gyakran Ismételt Kérdések) részt, ahol a látogatók konkrét segítséget kaphatnak a webhellyel kapcsolatos kérdésekben, vagy támogatást kaphatnak a technikai személyzettől e-mailben, illetve segítséget kérhetnek a szolgáltatással kapcsolatosan is.

Az utolsó frissítés részben az a dátum jelenik meg, amikor az oldalt utoljára frissítették, ez fontos információ lehet a látogatók számára. Ha a weboldal tartalmaz híreket, akkor az utolsó frissítés dátumából megállapítható, hogy az ott lévő tartalom aktuális-e még vagy már elavult.

4.3.4. Belső és külső honlapok

Számos vállalat tart fenn saját belső weblapot. Ezek olyan egyszerű oldalak gyűjteménye, amik egy helyi kiszolgáló portálon vagy az Intraneten vannak tárolva. Az ilyen webhelyek fenntartásának sok előnye van. A vállalat egyszerre sok dokumentumot tud tárolni egy helyen, és ezekhez biztonságos és gyors hozzáférést tud biztosítani minden alkalmazottja számára. Az eredeti dokumentumokhoz való hozzáférés könnyen vezérelhető a webmester, vagy egy felelős IT személy által.

Az itt tárolt dokumentumok sokfélék lehetnek. Ide tartoznak a vállalat különböző üzleti és szociális hirdetményei, mint például az állásajánlatok vagy promóciós lehetőségek, irányelvek, eljárások iránymutatói, nyomtatványok, letölthető és kitölthető űrlapok és számos egyéb dokumentumtípus.

Egyes vállalatok a belső sport és szociális klubok számára lehetőséget biztosítanak a weblapon, hogy az alkalmazottak számára tájékoztatást nyújtsanak a tevékenységeikről. Mások az étkezde menüjét osztják

meg, folyamatos frissítés mellett. Bizonyos szempontból az Intranet átvette a hagyományos hirdetőtáblák szerepét.

Ha nagy a vállalaton belüli mozgás, egy egyértelmű belső információs program segítségével az új emberek folyamatos képzésben részesülhetnek. Ha egy vállalat belső weblapot tart fent, lehetnek olyan esetek, amikor hasznos lehet, hogy az oldalakhoz olyan emberek is hozzáférhessenek, akik nem a cég alkalmazottai. Ezekben az esetekben korlátozott hozzáférés adható az oldal bizonyos részeihez a vevők vagy eladók számára. Az ilyen felosztás az úgynevezett Extranet. Példának tekintsünk egy céget, melynek rendszeres negyedéves rendelési előrejelzést küldenek az ügyfelei.

Valószínűleg ez a vállalat úgy dönt, hogy ezen ügyfelek részére, közvetlen hozzáférést biztosít az oldal egy adott részéhez, ahová a vevők közvetlenül a rendszerbe feltölthetik a saját igényeiket. Így a vállalatnak már nem kell külön felvenni a kapcsolatot azokkal az ügyfelekkel, akik ily módon rendelnek. Ezzel időt tud felszabadítani az ügyfélszolgálati alkalmazottak számára más feladatok elvégzésére.

4.3.5. Az üzleti honlap karbantartásának kihívásai

Mint bármely honlapon, a cégek honlapján is, ami belső használatra készül, vagy használják mind a vevők, mind az eladók, számos kihívással kell szembenézni. Az oldal tartalmának naprakésznek kel lennie, és arra is kell ügyelni, hogy az oldal mindig elérhető legyen a felhasználók számára.

Ezek közül a legnehezebb probléma az, hogy a tartalom könnyen frissíthető legyen. Pontosan meg kell határozni, hogy melyik tartalmat milyen gyakran kell frissíteni és hogy ez kinek a feladata. Sok cégnek nincs ebben tapasztalata a házon belül, sőt anyagi forrásai sincsenek meg ehhez.

Azok az oldalak, melyeknek gyakran frissítik a tartalmát, nagyobb valószínűséggel kerülnek előrébb a keresőkben, mint azok, amelyeket ritkán frissítenek.

A tartalom rendszeres frissítésének másik fő oka, hogy megtartsa a látogatókat. A tartalom a weboldal legfontosabb jellemzője, ez az elsődleges oka az oldal látogatottságának. Ha ezt a tartalmat nem frissítik, a látogatónak nincs miért visszatérni az oldalra az első látogatást követően. Azonban ha a tartalom hétről-hétre változik, az érdeklődők nagyobb valószínűséggel visszatérnek, hogy megnézzék, kínál-e valami újat az. Az ily módon megtartott ügyfelek bevétel növekedést jelentenek, pláne, ha az oldalon termékértékesítés is van.

Amikor a munkavállalók nem szívesen kezdenek el használni egy újonnan létrehozott belső webhelyet, el kell gondolkozni azon. hogy mi lehet ennek az oka. Lehet, hogy az oldal eredeti célja az, hogy fontos cégeljárásoknak és irányelveknek adjon helyet. Az is lehetséges, hogy szükség van más, az üzleti dolgokhoz kevésbé kapcsolódó tartalomra, annak érdekében, hogy a munkavállalók bejelentkezzenek az oldalra. Ennek egy lehetséges megoldása az, ha az oldalon, az alkalmazás használatához tippeket és trükköket helyezünk el, vagy akár versenyt is rendezhetünk, kis jutalmakkal a rendszeres felhasználók körében.

4.4. Webtervezés: követelmények és módszerek

Tanulási célok

- Felismeri a célcsoport(akik számára a weboldal készül) igényeit.
- Vázolja a weboldalon lévő túl sok információ hátulütőit.
- Vázolja a szerencsétlen színösszeállítással kapcsolatos problémákat.
- Bemutatja a felhasználóbarát honlapok fejlesztésének irányelveit, mint az olvashatóság, a kiemelt tartalom, a könnyű és következetes navigáció és a ”hol vagyok”.
- Bemutatja a webes szövegek általános minőségi szempontjait, mint a böngésző képességbeli problémái, a HTML-validálás, a tömör szöveges tartalom, a helyesírás-ellenőrzés és a kis bájtméretű grafikák.
- Kifejti a könnyű navigáció szükségét egy honlapon.
- Vázolja a honlap fejlesztésére alkalmas eszközöket.
- Vázolja a szerkezeti diagram célját és használatát a honlaptervezésben.
- Bemutatja a jelentősebb navigációs módszereket.
- Bemutatja a webtervezés néhány projektalapú megközelítését és általánosan használt technikáit, mint a forgatókönyv és a képernyőtervek.

4.4.1. Weblap tervezés: Kinek? Miért? Milyet?

A honlapkészítés legfontosabb része a koncepció és az elérendő célok pontos meghatározása. Annál is inkább, mivel az utólagos módosítások rengeteg plusz időt és pénzt igényelnek.

Először át kell gondolni és meg kell fogalmazni az összes olyan **célt**, amit a weblappal el szeretnénk érni. Ilyen célok lehetnek például a cég előnyös bemutatása, vagy tájékoztatás nyújtása a termékről, termékek értékesítése a weboldalon keresztül. Jó előre meghatározni például, hogy szeretnénk-e lehetővé tenni a termék adatlapjának letöltését és továbbítását. Vagy szeretnénk-e párbeszéded folytatni az ügyféllel a honlapon keresztül.

A második legfontosabb kérdés, hogy **kinek készül a weboldal**? Egy tinédzser egészen másképp gondolkozik, viselkedik az Interneten, mint az idősebb korosztály. Másképp kommunikál, másképp jut el hozzá az információ, másféle színek ragadják meg a figyelmét. Ezért minden célcsoportra különböző módon kell hatni, tehát az weblapot is ennek megfelelően kell kialakítani és felépíteni. Olyan weboldalt kell készíteni, ami tekintettel van a potenciális ügyfelek igényeire, elvárásaira és a megoldandó problémáira jó megoldást tud adni.

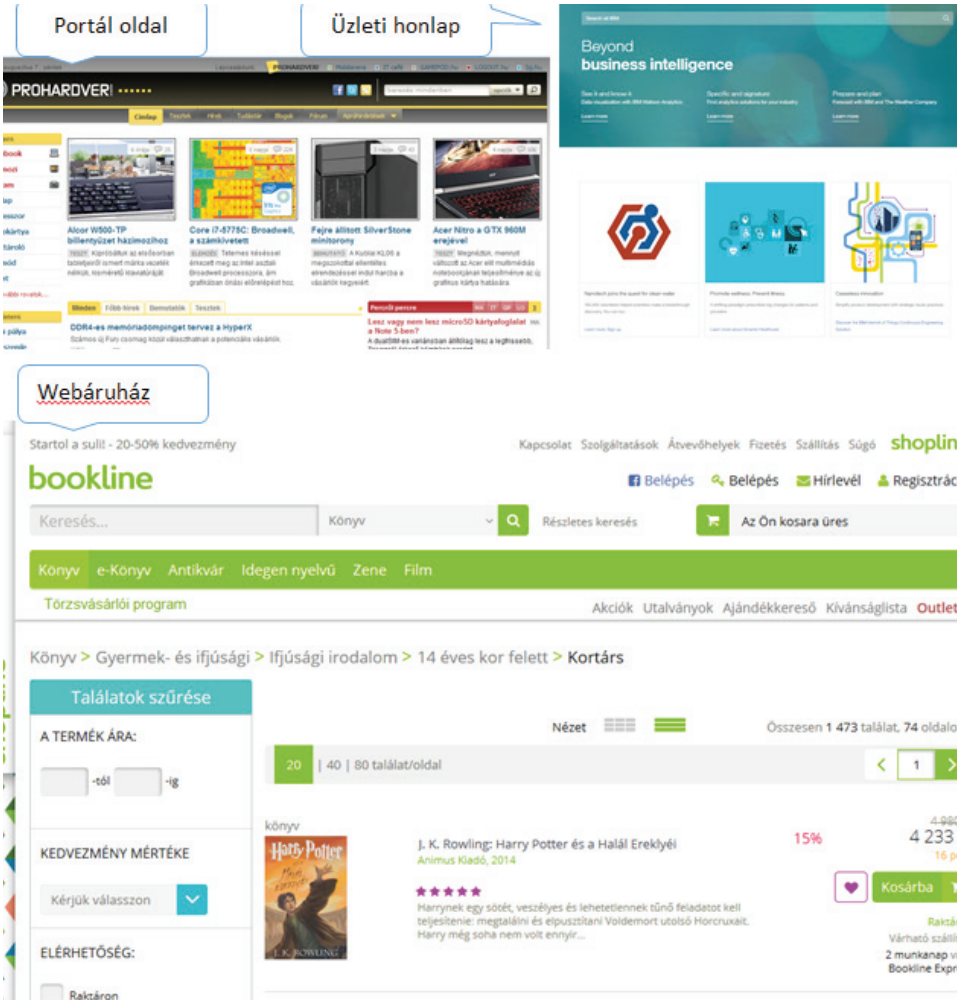
A következő kérdések közelebb vihetnek a célcsoport beazonosításához:

- Kiket érdekelhetnek a weboldalunkon található információk?
- Várhatóan milyen céllal érkeznek a látogatók az oldalra (vásárolni, információt gyűjteni, tájékozódni)?
- Mire lesznek kíváncsiak a látogatóink?
- Milyen műveletet szeretnének az oldal segítségével végrehajtani?
- Milyen a látogatók várható összetétele (versenytársak, média, vásárlók)?
- Várhatóan milyen demográfiai adatokkal jellemezhetők?
- Milyen az internetes kapcsolatuk (széles sáv/modem stb.)?
- Feltételezhetően milyen oldalakat ismerhetnek jól és használnak gyakran?

Weblapok típusai:

- Üzleti honlap: Egy cégről és annak termékeiről vagy szolgáltatásairól ad információt
- Információs weboldal: Egy bizonyos témáról (pl. gyereknevelés) tartalmaz bejegyzéseket.
- Portál, vagy híroldal: Egy témához kapcsolódó információk (pl. napi hírek, utazás) gyűjteménye, gyakran egy újság, tv csatorna számára készül. pl. az Index, Origo.
- Webáruház, webshop, hirdetési weboldalak: Célja a termékek vagy szolgáltatások eladása. A webáruházakban a termék kiválasztása, megrendelése és kifizetése a honlapon történik.

- Közösségi weboldal: Az emberek megoszthatnak egymással információkat, beszélgethetnek, kapcsolatokat alakíthatnak ki (Facebook, Twitter). Ilyen oldalon megjelenni egyre nagyobb üzleti értékkel bír.
- Személyes honlap: célja gyakran csak bemutatkozás a világnak, a barátoknak, többnyire nem üzleti céllal készül.
- Blog: A bejegyzések naplószerűen jelennek meg, sokszor személyes jellegű, de tartalmazhat közérdekű információkat is. Egy cég számára a rendszeres blogírás a marketing kiegészítő eleme lehet.



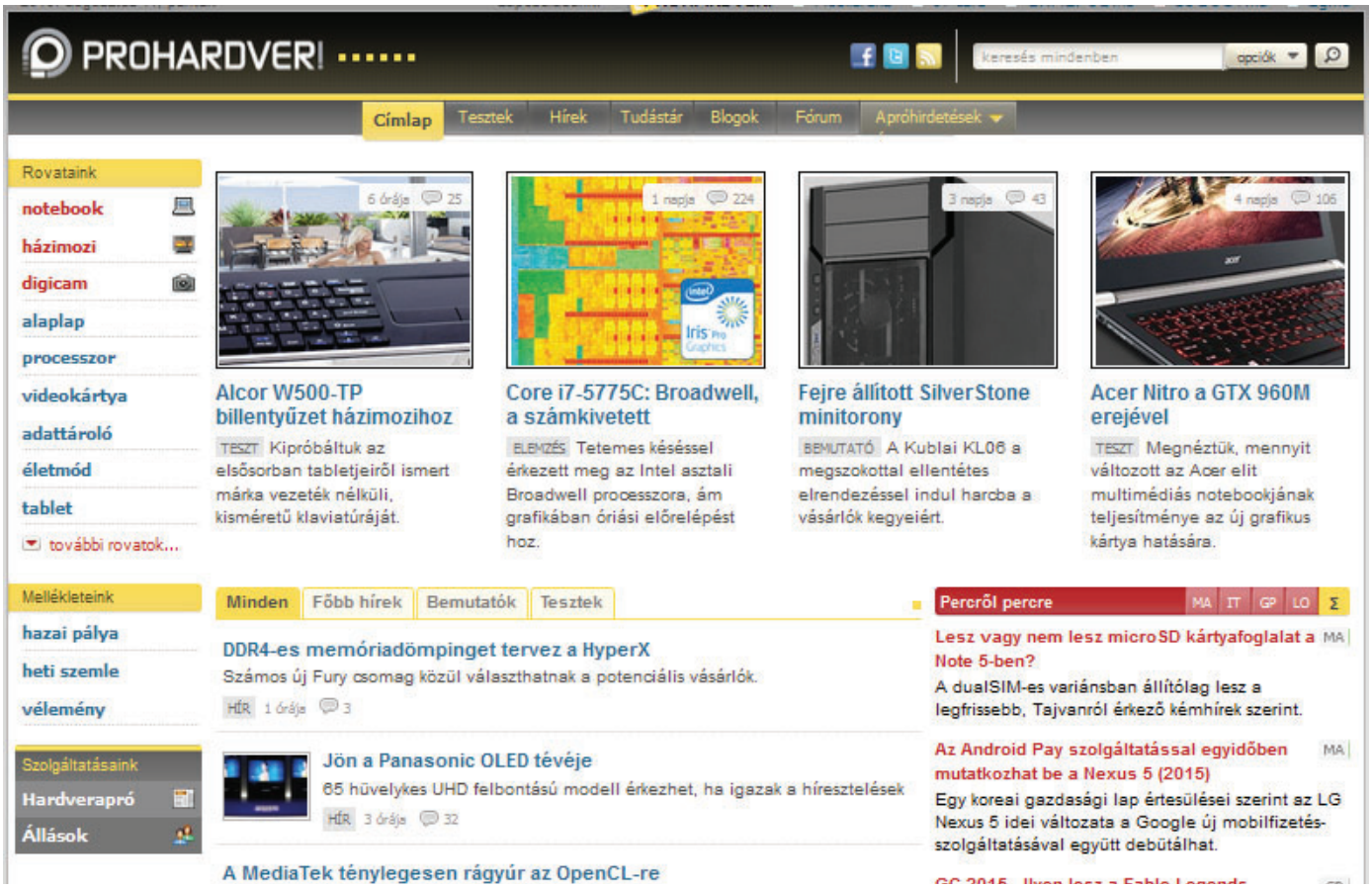
4.24. ábra Különböző típusú weblapok

4.4.2. Az információ mennyiségéről…

A látogatók nagy része valamilyen keresőmotor segítségével érkezik a weboldalra. Ha gyorsan megtalálja a keresett információt az oldalon, akkor ott is marad, ellenkező esetben visszakattint a keresőoldalra. Ezért fontos egy jó érkeztető oldal kialakítása.

A Kezdőlap egy érkeztető oldal, ahonnan mintegy további tájékoztató táblák mutatják az utat a további részletes információk felé. A látogatónak pontos képet kell kapnia arról, hogy mit tud csinálni a honlapon, hol találhatja meg az őt érdeklő információt. Ezért a Kezdőlapon csak annyi információt helyezzünk el, ami szükséges ahhoz, hogy a látogató felmérje, milyen további információkra van szüksége, és a megfelelő linkekre kattintva a kezdőlapról a számára értékes információval rendelkező tartalomhoz juthat.

A túl sok, vagy a nem megfelelően rendezett információ ahhoz is vezethet, hogy a látogató képtelen összehasonlítani és különbséget tenni a termékek között.



4.25. ábra Átlátható érkeztető oldal

Ha képeket is használunk, akkor ezt mértékkel kell tennünk, mert a túl sok kép összezavarja a látogatót, és fontos, hogy ne vonják el a látogató figyelmét zavaró reklám bannerek vagy felugró ablakok.

4.4.3. Még egyszer a színekről…

A színek szerepéről a 4.2.2. fejezetben már volt szó. Most a színek használatának azt az aspektusát vizsgáljuk, hogy mi befolyásolja a színek megjelenését.

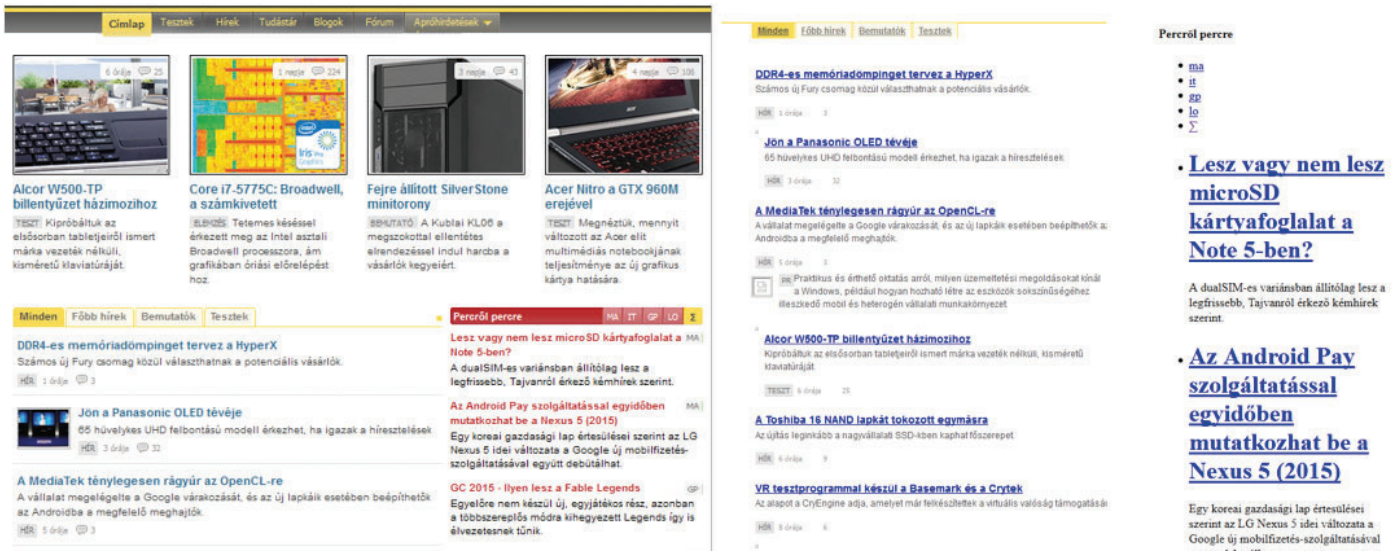
A színek érzékelését befolyásolják szubjektív tényezők, valamint az is, hogy milyen platformon fut a böngésző, milyen monitoron nézzük a weblapot, milyen a monitor beállítása. Épp ezért érdemes a weblapunkat a lehető legtöbb képernyőtípuson is tesztelni annak érdekében, hogy észrevegyük a színeltéréseket. Ha a szín nem jelenik meg megfelelően a felhasználó képernyőjén, akkor érdemes „web-biztos” színek használatával próbálkozni.

A weboldalakat úgy kell kialakítani, hogy színek nélkül is képes legyen minden információt ugyanúgy továbbítani, mint színesen. (például az összefüggésekben vagy kiemelésekben). Színvak, vagy színtévesztő felhasználók számára nagyon nehéz azoknak a honlapoknak a használata, ahol a színek használata az egyedüli módszert a képernyő elemek vagy vezérlők azonosítására.

A legegyszerűbben úgy próbálhatjuk ki, hogy vajon színek nélkül is átlátható-e az oldalunk, ha kinyomtatjuk fekete-fehérben. Ha ebben az esetben pl. nem látható a különbség a látogatott és nem látogatott linkek között, akkor javítani kell a színsémán.

A túl sok szín használatát mellőzni kell mind a háttér, mind a szöveg használata során. Legjobb elkerülni az össze nem illő színek együttes kombinációját, mint például a lila és sárga, mert a szöveg nehezen olvashatóvá válik.

Prohardver honlapja színekkel és színek/képek nélkül is jól használható: http://prohardver.hu/index.html



4.26. ábra Prohardver weboldala design elemekkel és a nélkül

4.4.4. Felhasználóbarát honlapok fejlesztésének irányelvei

A weblapnak tartalmaznia kell azt az információt, amit a felhasználó keres, és a felesleges információkat minimálisra kell csökkenteni, vagy el kell kerülni. A szövegnek könnyen olvashatónak kell lennie, nem tartalmazhat gépelési, helyesírási vagy nyelvtani hibákat. A teljes elérhetőség adatait (postázási cím, telefonszám, stb.) a látogató számára rendelkezésre kell bocsájtani egy kiemelkedő helyen az oldalon.

A grafikának a tartalom megértését kell segítenie, mintegy erősítve az oldal tartalmi mondanivalóját. A felesleges grafikát a letöltési idő minimalizálása érdekében kerülni kell. Minden oldalnak akkor is használhatónak kell lennie, ha a felhasználó böngészőjében *a kép megjelenítés* funkció a ki van kapcsolva.

4.27. ábra Ceres sütő weboldala képekkel és képek nélkül

A honlap tartalmát logikusan kell megszervezni, hogy a felhasználó könnyen és gyorsan megtalálja azt az információt, amit keres. A navigációs eszközöknek, mint például a nyomógomboknak, minden oldalon összhangban kell lennie a honlapon belül.

Az oldalnak jól kell kinéznie és jól kell működnie a főbb böngészők leggyakrabban használt verzióiban. Szükség esetén alternatív oldalt kell felajánlani azon felhasználó számára, akiknek a böngészője nem támogatja az újabb funkciókat.

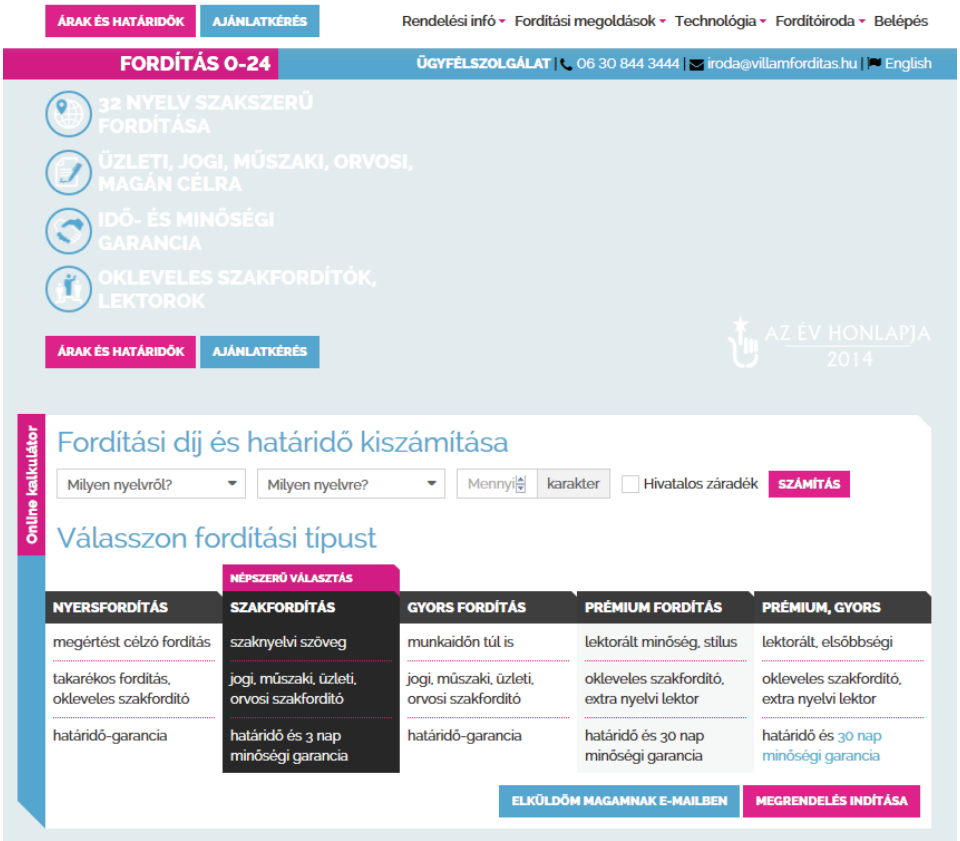
Ne legyen olyan oldal a weblapon, ami három kattintással nem érhető el bármely másik oldalról.

Az alábbi linken lévő videók bemutatják, milyen lenne egy rossz weblap a valóságban (angol nyelvű): http://hu.socialdaily.com/articles/2012-12-17/ilyen-lenne-egy-rossz-weboldal-a-valosagban

Példák 2009-ben készült legrosszabb weboldalakra: http://www.mrbottles.com/

4.28. ábra Példa rosszul felépített weblapra http://www.mrbottles.com/

Az Év Honlapja díjat nyert weblapok 2014-ben: https://www.azevhonlapja.hu/sajtohir/dijat-nyert-weboldalak-2014



4.29. ábra Példa jól felépített weblapra http://villamforditas.hu/

4.4.5. Webes tartalom minőségi szempontjai

A weboldal fejlesztésének fontos állomása az oldal majdani tartalmának összeállítása és annak témák szerinti rendezése. A tartalom csoportosítása hatással lesz a navigáció kialakítására. Ebben a lépésben már mindenképpen szükség van a majdani célcsoport bevonására.

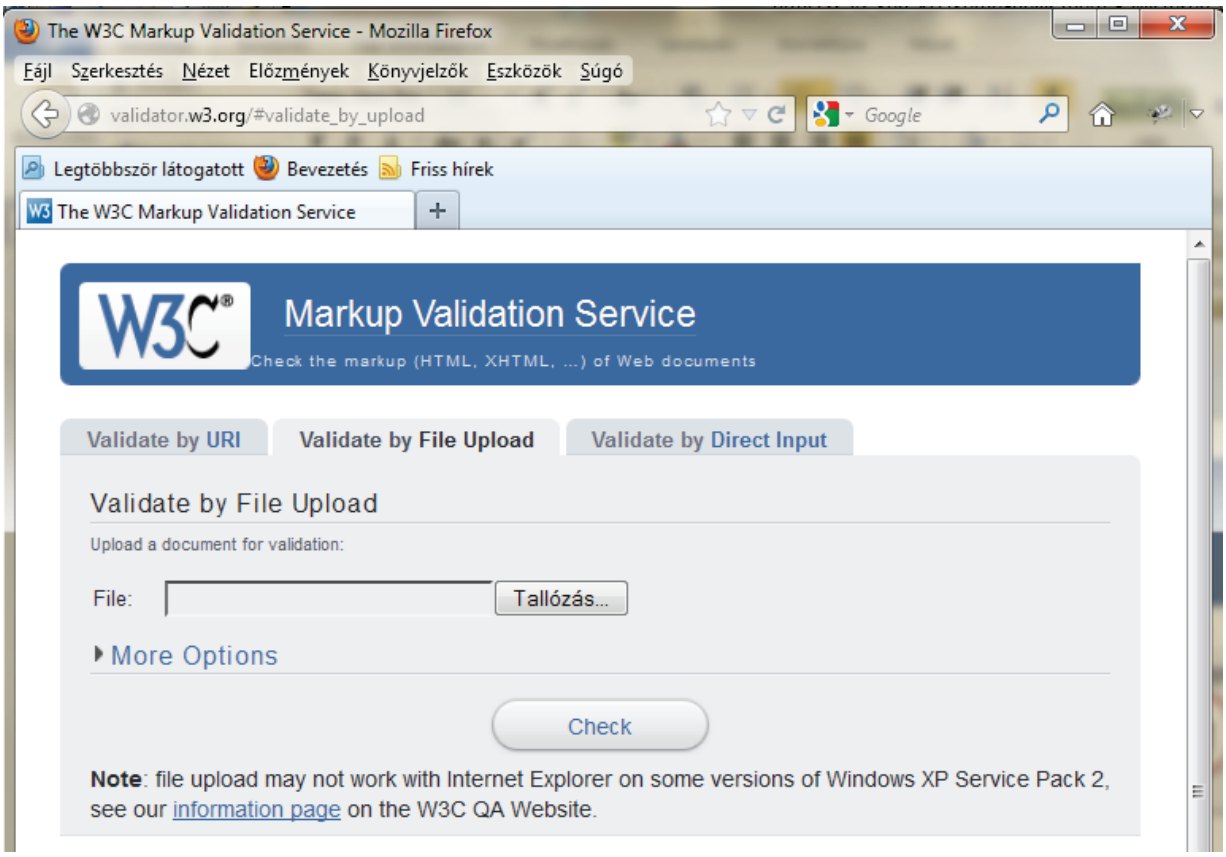
Amikor a weblap címét beírjuk a böngészőn címsorába, a weblap egy kérdést küld a böngészőnek. Lekéri a böngésző típusát, hogy ellenőrizhesse, hogy az adott böngésző támogatja-e azokat a funkciókat, amik szükségesek a weblap használatához. Ezek a funkciók lehetnek pl. a táblák, keretek, cookie-k, Java scriptek.

A weboldal nagyon eltérő módon jelenhet meg a különböző böngészőt használó felhasználók számára. Ezért fontos, hogy tervezéskor több böngészőben is ellenőrizzük a weblap kinézetét.

Egy jó honlapon nemcsak a helytelen írásmód van ellenőrizve, hanem a nyelvtani kifejezések helyessége is. A látogató nem lesz elragadtatva, ha egyszerű helyesírási hibákat talál az oldalon, és valószínűleg e miatt soha nem is tér oda vissza.

A grafikát a weboldalon általában navigációs gombok, logók vagy fotók esetében használunk, de lehetnek tartalmi funkció nélküli designelemek is. A grafikai elemeknek színben, betűtípusban, speciális effektekben egymással harmonizálniuk kell. Ügyelni kell arra, hogy gyorsan betölthetőek legyenek. Ennek az eszköze egy úgynevezett Optimalizáló, melynek segítségével minimalizálni lehet a grafikák méretét. Ha az oldal betöltése túl lassú, jó esély van arra, hogy a látogató nem fog várni, hanem elkattint onnan, még mielőtt az oldal teljesen betöltődne.

A tartalmi és a formai ellenőrzés után az oldal kódjának helyességét is ellenőrizni kell. A hibás kódú oldal ugyan megjelenik valahogy a böngészőben, de nem biztos, hogy pont úgy, ahogy mi szeretnénk. A hibás kód továbbá lassíthatja az oldal betöltődését, valamint a keresőmotorok is hátrább sorolják a találati listában. Fontos tehát, hogy az oldalunkat validáltassuk, azaz szintaktikai ellenőrzést végezzünk a kódon. Erre az Interneten elérhető validátorok segítségével van lehetőség (HTML validátor, CSS validátor, stb.)



4.30. ábra HTML validátor http://validator.w3.org/

4.4.6. Navigáció a honlapon

Navigáció szerepe az oldalon, hogy segítse a felhasználót abban, hogy egyik oldalról a másikra eljusson. Navigációnak teljesíteni kell az alábbi két legfontosabb funkciót:

- Egyrészt közölnie kell a felhasználókkal a számukra rendelkezésre álló lehetőségeket
- Másrészt biztosítani kell a mozgást számukra ezek között a területek között.

A navigációs linkek nagy szerepet játszanak abban, hogy mennyi időt tölt a látogató az oldalon, és hogy mennyire járja be a weblapot. A tiszta elrendezés, a rendezett navigáció és a megfelelő mennyiségű fehér terület a honlapon javítja az oldal kinézetét és használatát. Mindez segíti a felhasználót abban, hogy átlássa, pontosan mi áll rendelkezésre az oldalon, és segít megtalálni azt az információt, amit keres.

Fontos, hogy egységes legyen minden oldal megjelenése, és csak a tartalom változzon az oldalak között. Nagyon zavaróak a látogató számára a honlap egyes részei között az eltérő navigációs stratégiák.

Használjunk rövid, világos és pontos szavakat a menükben, hogy a látogatók tudhassák, mit tartalmaz a megfelelő oldal.

Ahol lehet, használjuk az egyezményes ikonokat, ne próbáljon meg olyan saját egyedi ikonokat fejleszteni, amit a látogató nem ismer fel. Használjunk könnyen érthető és azonnal felismerhető szabványos ikonokat.

Fontos, hogy a látogató mindig vissza tudjon térni a már meglátogatott oldalakhoz.

Tartalmazzon Keresés funkciót, hogy a látogató könnyen meg tudja keresni azt, ami érdekli.

4.4.7. Honlap fejlesztésének eszközei

A weboldalakat létrehozhatjuk, úgy hogy mindent HTML-kódban írunk meg, de ez így egy nagyon hosszadalmas munkafolyamat, arról nem is beszélve, hogy az apró hibákat a kódban nehéz megtalálni és kijavítani. Sok, a honlapkészítést megkönnyítő eszköz áll a rendelkezésünkre az egyszerű szövegszerkesztőtől kezdve a számos multimédiás funkció hozzáadására képes fejlett szoftvercsomagokig.

Fejlesztő eszközök:

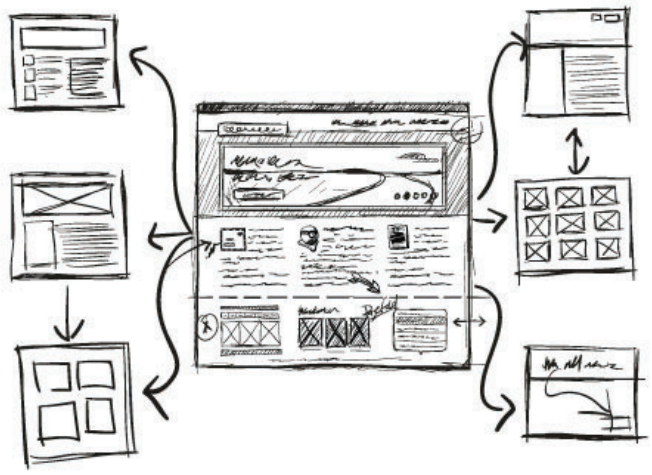
- Egyszerű szövegszerkesztő, a forráskód szerkesztésére. Szerencsés, ha ismeri az adott nyelv (HTML, JavaScript stb..) elemeit, ezeket többnyire automatikusan eltérő színnel emeli ki. Pl: PsPAd, Notepad++ stb.
- Vizuális szerkesztők, melyekkel grafikus felületen állítható össze a honlap, akár egyetlen kódsor írása nélkül is. Pl. Dreamweaver.
- Különböző képszerkesztő és képátalakító programok. Pl.: Adobe Photoshop, Gimp, Flash, GofBot, Irfanview
- A programozási nyelvekbe beépített kisalkalmazások, játékok, programozási útmutatók.
- Fájl feltöltő szoftver: a WS_FTP használatával tudják az elkészült honlapot a fejlesztők feltölteni egy szerver tárhelyére, úgy hogy az elérhető legyen a weben.
- Az oldal kiértékelő szoftver: Elvégzi az oldal kódjának szintaktikai ellenőrzését, méri az oldal letöltési idejét, javaslatot tesz a képek méretének optimalizálására, ellenőrzi a böngésző kompatibilitást, megkeresi és kijavítja a hibás linket. Pl.: Net Mechanic.

4.4.8. A weblaptervezés folyamata

A weboldalak készítése során a teljes fejlesztési folyamatot érdemes előre megtervezni. E célból fejlesztési terveket szokás készíteni. Ezek tartalmazzák a részletes feladatlistát, a felelősöket, mi élvez prioritást a fejlesztés során, tartalmazza az ütemezést, a mérföldköveket és a költségkeretet.

A tervezési szakaszban elemezzük a potenciális versenytársak weboldalait, átgondoljuk a várható látogatói forgalmat, és eldöntjük, hogy az oldal hatékonyságának mérésére milyen eszközt, módszert használunk majd.

A szerkezeti diagramok segítenek a tervezői csapat tagjainak az alkalmazás szerkezetéről tárgyalni, és ennek segítségével mindenki a legfontosabb részre, a tervezésre tud koncentrálni.



4.31. ábra Weblap szerkezeti diagram

Ezután elkészül a majdani honlap sematikus ábrája, a drótváz, vagy wireframe. A drótváz is grafikusan jeleníti meg a készülő honlapot, de még a design elemek nélkül. Célja a weboldal arányainak megtervezése, funkcióinak beillesztése és elhelyezése. A wireframe grafika elkészítése előtt már kész oldalstruktúrával és funkciókínálattal rendelkezünk a készülő lap kapcsán, így a sematikus ábra már a menüpontokat és szolgáltatásokat is képes megmutatni a megrendelő számára.

4.32. ábra Drótváz

Miután összeállt az egész tervezési folyamat, elkészült a tartalom logikus csoportosítása, megtervezhetjük a navigációt, és az oldalak felépítését, elkészülhetnek a grafikai tervek. Legalább 3 féle grafikai tervet szokás készíteni, ezek közül választhat a felhasználó. Ezután megkezdődhet a weblap elkészítése a tervek szerint.

Az weblapok készülhetnek előre gyártott sablonok alapján is. A weblapsablon egy minta weboldal kész designnal, navigációval, szabványoknak megfelelő weboldal, csak tartalommal kell megtölteni. Ezzel a módszerrel költséghatékonyan készülhet el céges weboldal, viszont kevésbé testre szabható.

Ennél talán jobb megoldás lehet, ha valamilyen tartalomkezelő rendszert (Joomla) használunk a honlapunk elkészítéséhez. Rengeteg beépülő modul segítségével személyre szabhatjuk oldalunkat. Bár a designhoz sablonalapú megoldásokat használnak ezek a rendszerek is, de az egyes grafikai elemek széles körben személyre szabhatóak.

A weblapkészítés során folyamatosan tesztelni kell a weboldalt a leendő felhasználókkal és a visszajelzéseik alapján érdemes még a fejlesztési folyamat során beépíteni a módosításokat.

Az is fontos szempont, hogy célközönség könnyen megtalálja az oldalt, keresőrobotok (Google) találati listájukban lehetőleg az elsők közé sorolják. Ezért fontos a tervezés során már keresőmotor optimalizálási technikák használata.

4.4.9. Navigációs módszerek

Jó tervezési stratégia, ha a honlap minden oldalán elhelyezünk egy **Kezdőlap** linket, mivel ez lehetővé teszi a látogatóknak, hogy visszataláljanak a főoldalra és onnan kiindulva keressék meg a honlap más területeit.

4.33. ábra Navigációt lehetővé tévő helyek a weblapon

A **beágyazott linkek** a navigáció legalapvetőbb formái. Ezek lehetnek szövegesek vagy grafikusak.

A szöveges linkek sokat segítenek azoknak, akik valamilyen felolvasó szoftvert használnak, vagy gyakran nézik az oldalt kikapcsolt képmegjelenítéssel, vagy újak az Interneten. Ha grafikus ikonokat vagy képeket használunk, azok legyenek mindig érthetőek. Az ikonoknak az a célja, hogy megkönnyítsék azoknak az embereknek a dolgát, akik nem beszélik a honlapon használt nyelvet. Nyilvánvaló azonban, hogy egy olyan ikon, amiről nem állapítható meg egyértelműen, hogy mit takar, rosszabb, mintha egyáltalán nem volna.

Az **morzsamenü** nagy mennyiségű információ szervezésénél hasznosak. Ezek olyan szöveges linkek vízszintes sorozata, melyek megmutatják az adott oldal elérési útvonalát a kezdőlaptól kiindulva. Ezek a linkek információt tartalmaznak arról, hogy az oldalak milyen hierarchikus kapcsolatban állnak egymással.

A **navigációs sávok** nagyon gyakoriak és általában véve jól használhatónak bizonyultak. A lap tetejére vagy oldalára szokták helyezni őket. Nagy előnye, hogy könnyen használhatóak, és minden ember képes használni, az is, aki csupán néhány percet töltött el az Interneten. Hátránya, hogy értékes helyet vehet el, amit a tartalomra jobban fel lehetne használni. A navigációs sávnak tartalmaznia kell a honlap összes fő részét, és minden oldalon egységesen ugyanúgy kell kinéznie.

A navigációs sávnak minden oldal szerves részeként kell jelen lennie, és elrendezésében összhangban kell lennie a többi oldallal.

4.4.10. A webtervezés projektalapú megközelítése

Mivel a weboldalak különböző típusúak és méretűek, nem lehet ugyanúgy tervezni őket.

A weboldalak tervezése attól függ, hogy az egy üzleti weblap belső céges oldala az alkalmazottak számára vagy csak egy klub első kísérlete arra, hogy jelen legyen a Weben. Lehet a weboldal célja a szórakoztatás, a tájékoztatás, grafikák megjelenítése, vagy egy egyedülálló szolgáltatás nyújtása.

Ha ez egy üzleti oldal, lehet az a célja, hogy új ügyfeleket szerezzen, információt nyújtson a termékekről és szolgáltatásokról, piackutatást végezzen, vagy segítséget nyújtson ügyfelei számára.

A jó forgatókönyv-vázlatok segítségével könnyedén megérthető egy weboldal szerkezete, átlátható, hogyan és milyen információt fog majd bemutatni. A forgatókönyv tehát a honlap formájának és tartalmának koncepcionális vázlata.

A forgatókönyvek előnyei közé tartozik:

- Az a képesség, hogy megossza az oldal szerkezetét a tervezők csapatával.
- Egy egyértelmű, kézzelfogható bemenetet biztosít a tervezéshez és fejlesztéshez.
- Lehetőséget teremt arra, hogy még a fejlesztés és a költségek tervezését megelőzően átlássuk, hogyan lesz a weboldal szervezett és a látogatók által navigálható
- Az információ felépítésének szerkezetéből rájöhetünk az esetleges problémákra, ezáltal javíthatjuk is azokat az éles használat előtt. Így elkerülhetjük, hogy a keresőmotorok a hibás oldalunkat indexeljék, ezáltal mintegy archiválva a hibát.
- Segítségével még idejében tesztelhetjük, hogy mennyire felhasználóbarát az oldalunk.

Egy tipikus internetes oldal forgatókönyv-vázlata három részre osztható:

- A weboldal egy gyors vázlata, szerkezete: ez fogja megmutatni mindegyik oldalt és a címét.
- Egy részletes szerkezeti vázlat: mindegyik oldal internetcímét felsorolja egy szervezett formátumban.
- Mindegyik oldalnak egy részletes vázlata: mindegyik oldalt kézzel vagy egy oldalelrendezést bemutató programmal, minden szöveget, grafikát és hiperhivatkozást részletesen vázol.

Weboldal vázlatok létrehozásának az az előnye, hogy még a közzététel előtt kiadható a végfelhasználónak tesztelés céljából. Ezáltal vizsgálhatóvá válik, hogy a kapcsolatok mindig megfelelően működnek-e. Lehetőséget ad arra is, hogy felülvizsgáljuk, módosítsuk a helyesírási hibákat az egész oldalon. Még idejekorán tesztelhetjük, hogy minden kapcsolat az összes külső oldalakon úgy működik-e, mint ahogy tervezték, és ezáltal időt takaríthatunk meg, vagyis több idő marad bármilyen multimédiás tartalom előkészítésére, valamint a fogadó szerverre való feltöltésre.

4.5. Weblapok kialakítása

Tanulási célok

- Felvázolja a jelölő nyelv fogalmát és bemutatja a HTML főbb jellemzőit.
- Használja az HTML alapparancsokat, és értelmezi az elrendezésre (layout) vonatkozó parancsokat, mint a kemény és a lágy formátum, a speciális karakterek, a szövegtagolók, az igazítás, a fejlécek, a kép tagek, a hátterek, a színek, a linkek, a listák, a táblázatok, az űrlapok és a keretek.
- Felvázolja az írott szövegre vonatkozó alapvető grafikai szabályokat, mint a betűméret és a százalékosan megadott közök (sorköz, térköz, betűköz).
- Felvázolja az XML alapelemeit és alkalmazásait, a fejlődési utat a HTML-től az XHTML-ig.
- Bemutatja a stíluslapok fogalmát, mint a CSS (Cascading Style Shheets) és az XSL (Extensible Stylesheet Language). Érti a kialakításban való használatukat.

4.5.1. A jelölő nyelv fogalma – a HTML jellemzői

A jelölő nyelvek utasításai megmondják az értelmező programok számára, hogy az utasítások közé zárt szöveget miképp kell értelmezni, felépíteni, formázni, vagy feldolgozni, illetve mi az egyes szövegelemek szerepe. A nyelv jellemzője, hogy mind az utasításokat, mind az utasításokhoz tartozó tartalmakat azonos módon, szövegesen írja le. A jelölő nyelv tehát lehetőséget nyújt arra, hogy egy szövegalapú információ szerkezetét is ugyanabban a dokumnetumban tároljuk.

Mindezt úgy éri el, hogy bizonyos szöveget hivatkozásként jelöl, más szöveget címsorként, vagy bekezdésként, listának stb. A szöveget ki lehet egészíteni interaktív űrlapokkal, beágyazott képekkel, és más objetumokkal, melyek elhelyezésére szolgáló utasításokat szintén szövegesen adunk meg az oldalon.

A számítástechnikában ma használt egyik legismertebb leíró nyelv a Hyper Text Markup Language (HTML), a World Wide Web protokolljai közül az egyik, mely a weblapok lerását teszi lehetővé

A weboldalak nem mások, mint HTML-parancsokat tartalmazó egyszerű szöveges fájlok, amit a böngésző értelmez, és ennek megfelelően jeleníti meg a tartalmat az oldalon. A nyelv a hipertext elvei szerint működik, azaz a szövegben hivatkozásokat helyezhetünk el, ami lehetővé teszi a weblap nem lineáris bejárását.

Weboldalak HTML-tagek használatával épülnek fel. A tagek a jelölők a HTML dokumentumban. Ezeket a jelölőket használják, hogy meghatározzák, hol kezdődik egy elem és hol ér véget.

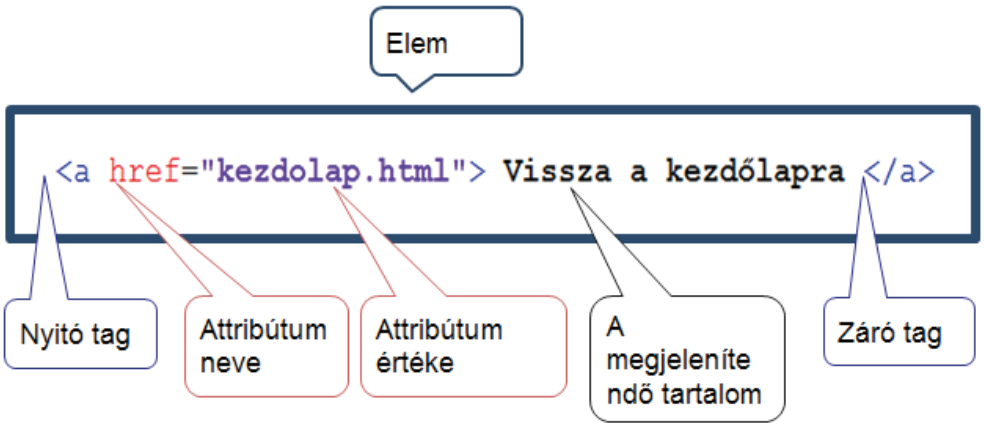
Egy HTML-elem felépítése:

A html-elemek három részből épülnek fel:

- címkék, vagy tag-ek,
- attribútumok,
- és a megjelenítendő tartalom.

A tag-ek a böngészőnek szóló megjelenítési utasítások. A tartalmakat nyitó és záró tag-ek között helyezzük el. A tag-ek közötti szöveg jelenik meg a böngészőben. Egy tag tehát a kisebb (<) és a nagyobb (>) jelek között áll. A záró tag-et úgy tudjuk jelezni, hogy rögtön a kezdő < jel után egy / (perjelet) teszünk. Minden egyes tag-et le kell zárni.

Egy HTML utasítás tehát az alábbi módon néz ki:



4.34. ábra HTML elem felépítése

<tag attribútum1="érték" attribútum2="érték">tartalom</tag>

Az attribútum áll az attribútum nevéből, ez most a href, és az értékéből, a név és az érték közé egyenlőségjelet kell tenni. Az attribútum értékét pedig írógépi idézőjelek („ „) közé kell írni. A címkék és az attribútumok neveit kisbetűvel kell írni!

A HTML nyelvben a tagek határozzák meg az egyes szövegelemek funkcióit az oldalon. Kialakíthatunk címeket, alcímeket, bekezdéseket, felsorolásokat, táblázatokat, kereteket, megjelölhetünk idézeteket, vagy egyéb, jól meghatározott funkcióval rendelkező elemeket. Ezeket az egységeket CSS segítségével formázhatjuk a szövegszerkesztéseben megismert formázási lehetőségekkel.

4.5.2. Leggyakoribb HTML elemek

4.35. ábra A HTML oldal kötelező elemei

Az első sor az úgynevezett dokumentumtípus-meghatározás. Tudatjuk a böngészővel, hogy melyik HTML verzió nyelvi elemeit használtuk az oldal készítése során.

Az oldalunkat mindig <html></html> tagek közé kell írni.

Ezután következik dokumentumunk fejléce a <head>...</head> címkék között, ami nem jelenik meg a böngészőben, de fontos adatokat tartalmaz a dokumentum egészére vonatkozólag.

Az első sor a karakterkódolást, a második a weboldal címét határozza meg.

A <title> tag között lévő szöveg fog megjelenni a böngészőablak címsorában.

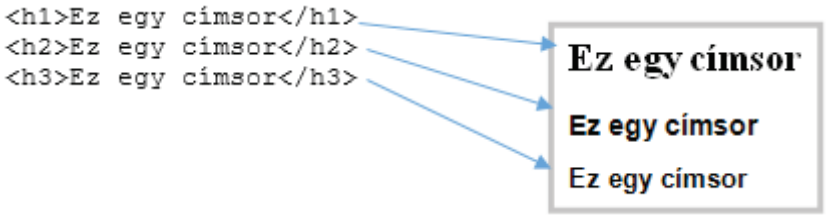
A <body>...</body> címkék közé kell írni az egész weboldal tartalmát.

Arra figyeljünk, hogy ide nem szabad azonnal szöveget írni, mert ezt nem engedi meg a szabvány, hanem például egy bekezdést kell definiálni, és oda már írhatunk szöveget. De legegyszerűbb, ha a div címke általános tároló elem közé kerül a weboldal tartalma:

Ez a lehető legegyszerűbb HTML dokumentum, minden új weboldal készítésekor szóról szóra ezzel a szöveggel kell kezdenünk.

Címsorok és bekezdések

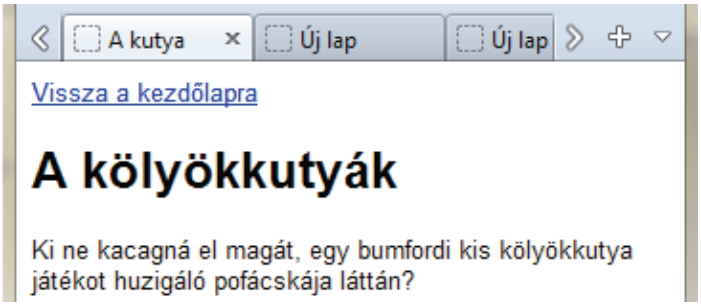
A címsorokat a <h1> ... <h6> címkék definiálják, a <h1> jelenti a legnagyobbat, a <h6> a legkisebbet. Általában csak <h3>-ig használjuk a címsorokat. Minden címsor elé és mögé automatikusan odakerül egy üres sor.



Bekezdéseket a <p> címkével definiálunk. Automatikusan elé és mögé kerül egy üres sor.

```
<a href="kezdolap.html"> Vissza a kezdőlapra </a>
<h1>A kölyökkutyák</h1>
<p> Ki ne kacagná el magát, egy bumfordi kis kölyökkutya játékot huzigáló
pofácskája láttán?</p>

<img src="kiskutya.jpg" alt="A kölyökkutyát rengeteget játszanak" width="320px"
height="262px" title="Kölyökkutya" />
```



Számozatlan lista

A számozatlan lista egy felsorolás, minden listaelem előtt egy kis pöttyel. Számozatlan listát az <ul> (unordered list, azaz számozatlan lista) címke vezet be, minden listaelem a <li> (list element, azaz listaelem) és a címkék között van. A listaelemeken belül írhatunk sortörést, bekezdéseket, képeket stb.

```
<ul>
      <li>Marmagasság: 35-39,5cm</li>
      <li>Testtömeg: 7-8kg</li>
      <li>Várható élettartam: 13-14 év</li>
</ul>
```

- Marmagasság: 35-39,5cm
- Testtömeg: 7-8kg
- Várható élettartam: 13-14 év

Számozott lista

Számozott listát az <ol> (ordered list, azaz számozott lista) címke vezet be, az elemeket itt is ugyanúgy kell megadni. A listaelemeken belül írhatunk sortörést, bekezdéseket, képeket stb.

```
<ul>
      <li>Marmagasság: 35-39,5cm</li>
      <li>Testtömeg: 7-8kg</li>
      <li>Várható élettartam: 13-14 év</li>
</ul>
```

1. Marmagasság: 35-39,5cm
2. Testtömeg: 7-8kg
3. Várható élettartam: 13-14 év

Hivatkozások készítése

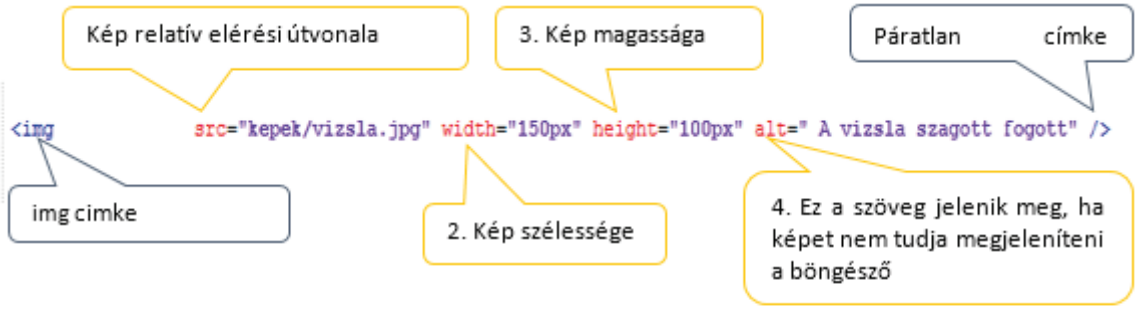
Az <a>... címkék közé írt szöveg linkként fog viselkedni, azaz a böngésző kék színnel aláhúzza, és a rajta való kattintáskor egy másik weboldalra tudunk eljutni. Azonban ehhez azt is meg kell mondanunk, hogy pontosan melyik weboldalra mutasson a link, különben honnan tudná a böngésző, hogy melyik oldalt kell betöltenie? Erre való a href nevű attribútum. Az attribútumokat mindig a kezdő címkébe kell írni, ha több attribútum van, akkor szóközzel választjuk el őket egymástól.

Az attribútum áll a nevéből, ez most a href, és az értékéből, a név és az érték közé egyenlőségjelet kell tenni. Az attribútum értékét pedig írógépi idézőjelek („ „) közé kell írni.
<címke attribútum1=”érték” attribútum2=”érték”> tartalom </címke>

A címkék és az attribútumok neveit kisbetűvel kell írni!

Képek elhelyezése a weblapon

Képet a következő módon szúrhatunk be:



`<img src:` A kép fájlneve, és elérési útvonala

`width` és `height:` A kép szélessége és magassága pixelben

`alt:` Ez a szöveg fog megjelenni, ha akármilyen okból nem jeleníthető meg a kép

Táblázatok

A táblázatot a <table>...</table> címkék közé kell írni. A táblázat egy sora a <tr>...</tr>, ezen belül pedig a cellák a <td>...</td> címkék közé kerülnek.

```
<table>
  <tr>
    <td>első sor, első oszlop</td>
    <td>első sor, második oszlop</td>
    <td>első sor, harmadik oszlop</td>
  </tr>
  <tr>
    <td>második sor, első oszlop</td>
    <td>második sor, második oszlop</td>
    <td>második sor, harmadik oszlop</td>
  </tr>
  <tr>
    <td>harmadik sor, első oszlop</td>
    <td>harmadik sor, második oszlop</td>
    <td>harmadik sor, harmadik oszlop</td>
  </tr>
</table>
```

első sor, első oszlop	első sor, második oszlop	első sor, harmadik oszlop
második sor, első oszlop	második sor, második oszlop	második sor, harmadik oszlop
harmadik sor, első oszlop	harmadik sor, második oszlop	harmadik sor, harmadik oszlop

4.5.3. A szöveg megjelenése

Weblapunkon a tartalom nagy része szöveges formában lesz elérhető. Ezért fontos, hogy a gondosan tervezzük meg a szöveges tartalmak megjelenését, valamint gondoljuk át, milyen információkat szeretnénk kiemelni és átadni látogatóinknak. Fontos, hogy pontosan fogalmazzuk meg a szöveget, ne legyenek benne helyesírási, stilisztikai hibák sem, valamint hogy képileg is megfelően helyezkedjenek el az oldalon. A betűtípusok, színek, valamint a szövegelemek tagolásával könnyen olvashatóvá tehetjük az oldalunkat, és a megfelelő helyre terelhetjük a látogató figyelmét.

Tipográfia a weboldalon

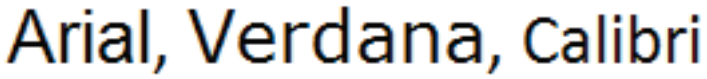
A tipográfia a szövegek megjelenése, dizájnja és elrendezése. Alapvetően négyféle betűtípust különböztetünk meg:

Talpas vagy serif: minden olyan betűtípus, amely egy befejezett vonást, szélesedő vagy vékonyodó vonalvégeket tartalmaz, vagy talpas végei vannak (beleértve a lapos talpakat is), ebbe a családba tartozik. Ez a betűtípus inkább nyomtatásban használatos,képernyőn nehezen olvasható.

Times Nem Roman, Garamond, Cambria

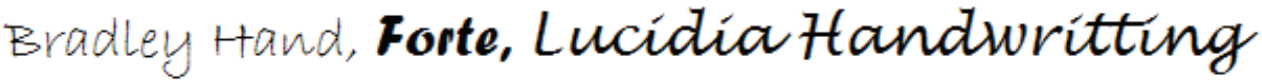
4.36. ábra Talpas betűtípusok

Talpatlan vagy sans-serif: minden olyan betűtípus, amelyben a vonásoknak egyszerű, sima végeik vannak, nem vékonyodnak vagy vastagodnak el, nincsenek benne talpak vagy más díszítések, azok ebbe a családba tartoznak. Képernyőn könnyen olvasható betűtípus



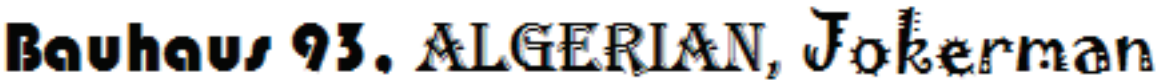
4.37. ábra Talpatlan betűtípusok

Írott vagy kurzív: ezek a betűtípusok általában egy kézzel írott, tollal vagy ecsettel készített betűkre hasonlítanak, és nem a nyomtatott betűkre. Weboldalakon ritkán használjuk, inkább csak rövid szövegelemek esetében, amikor ki akarunk emelni valamit.



4.38. ábra Kézírást utánzó betűtípusok

Díszítő betűk: Az ilyen betűkészleteket általában díszítő elemként használják. Nagyon látványosak, általában nehezen olvashatók. Főcímek esetén használatosak.



4.39. ábra Díszítő betűk

A weblapunk minden oldalán azonos betűtípusokat használjunk. Alapvetően 2-3 betűtípusnál ne használjunk többet. Fontos kiválasztani az összeillő betűtípusokat. A címek, címsorok betűtípusai lehetnek nagyobb betűméretűek, használhatunk talpas, vagy valamilyen díszítő betűt. A folyó szöveget azonban mindig jól olvasható talpatlan betűtípussal írjuk. Elsődleges szempont a jó olvashatóság.

A kontraszt

Akkor beszélünk kontrasztól, ha két elem között nagy a különbség, valamilyen formajegy tekintetében élesen eltér egymástól. Minél nagyobb a különbség az elemek között, annál nagyobb a kontraszt. A kontraszt létrehozásának általános módszerei, ha különböző betűméretet, színt, típust és nagy szótávolságokat alkalmazunk.

Egy oldalon kontraszt nélkül az olvasó nem tudja mit nézzen meg először, mi az, ami fontos. A kontraszt érdekesebbé teszi az oldalt, úgy, hogy az olvasó könnyebben felfigyeljen arra, hogy mi van az oldalon. Segíti az olvashatóságot, kiemeli a címlapokat és az alfejezeteket. A kontraszt felhívja a figyelmet a fontos részekre azáltal, hogy a kisebb vagy világosabb elemek visszahúzodnak az oldalon, hogy más elemek a középpontba kerüljenek.

4.40. ábra Kontraszt és tipográfia egy weblapon

Fehér foltok

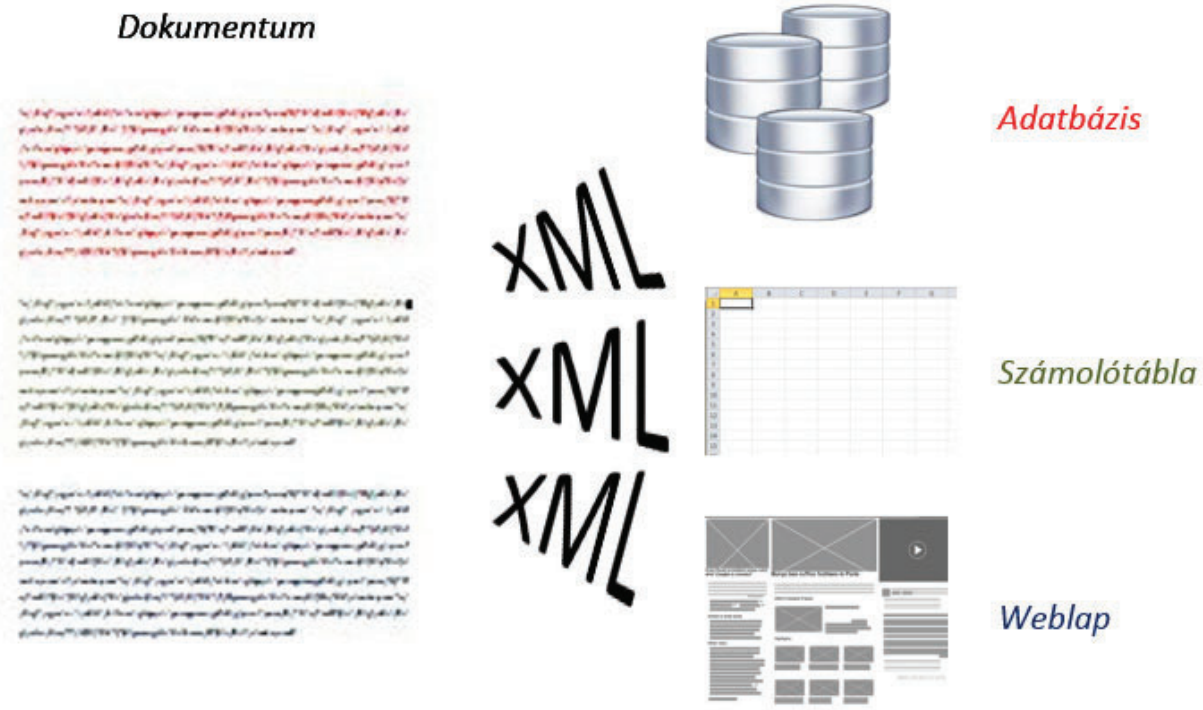
A szóközhasználat a dizájntervnek nagyon hatásos eleme. A szóköz nem egyenlő a semmivel. A szóköz, vagy a fehér területek az oldalon a szöveg és grafikák távolléte. Ez tagolja a szöveget és grafikákat. Vizuálisan jelzi a levegővétel helyét, irányítja a figyelmet

A bekezdések közötti hely növelését érhetjük el, ha:

- növeljük az oszlopok közötti teret.
- növeljük a sorok közötti távolságot.
- növeljük a címsorok és alcímek közötti helyet.
- több tér szabadon hagyása az oldalak széle körül szélesebb margók hozzáadásával.
- növeljük képi elemek körüli teret.

4.5.4. Az XML alapelemei és alkalmazásai, az XHTML nyelv

Az XML a EXtensible Markup Language (*Kiterjeszthető Jelölő Nyelv)* rövidítése. Ez egy leíró nyelv, ami nagyon hasonlít a HTML-re. Adatok átvitelére tervezték, nem adatok megjelenítésére. Itt a tagek nem előre definiáltak, saját magunk hozhatunk létre beszédes, ember számára könnyen értelmezhető címkéket. Míg a HTML célja az adatok megjelenítése volt, addig az XML az adatok leírása, tárolása és továbbítása. Egy jó XML leírás szoftver- és hardverfüggetlen információ szállító eszköz.



4.41. ábra XML kimenetek

XML –t gyakran használnak az internetes fejlesztések során, mivel sok szempontból leegyszerűsíti az adatok tárolását és megosztását. Ha dinamikus adatokat kell a HTML dokumentumban megjeleníteni, akkor nagyon sok a munka, valahányszor adatváltozás történik a HTML-kódot módosítani kell.

Az XML adatok külön XML-fájlokban tárolhatók.

JavaScript segítségével el tudjuk olvasni a külső XML fájlt, és frissíteni tudjuk a HTML adattartalmát.

Az XML adatokat egyszerű szöveges formátumban tárolják. Ez sokkal könnyebbé teszi, ha olyan adatokat hozzunk létre, amiket a különböző alkalmazások megoszthatnak egymással. Az adatok könnyen cserélhetőkké válnak összeférhetetlen rendszerek között. Ez szintén megkönnyíti egy új operációs rendszernek, egy új alkalmazásnak vagy egy új böngészőnek az adatvesztés nélküli bővítését vagy frissítését.

```
<?xml version="1.0" encoding="UTF-8"?>
<cimjegyzek>
        <ismeros>
                <nev>Kala Pál</nev>
                <tel>06-1-2345678</tel>
         </ismeros>
        <ismeros>
                <nev>Lement Elek</nev>
                <tel>06-1-87654321</tel>
         </ismeros>
</cimjegyzek>
```

4.42. ábra XML fájl struktúra

- Az XML dokumentum egyetlen gyökérelemet tartalmaz, ezen gyökérelem közé írjuk az XML további elemeit.
- Minden elem rendelkezik egy nyitó és egy záró tag-gel; ezek a tagek kis – és nagybetűérzékenyek.
- Egy elem tartalmazhat másik elemet, egyszerű szöveget vagy a kettő kombinációját.
- Az attribútumok segítségével meghatározhatjuk a tárolandó adat típusát is.
- Az tagek tartalmazhatnak attribútumokat, de követni kell az alábbi elnevezési szabályokat:
- A nevek tartalmazhatnak betűket, számokat, és más karaktereket.
- A nevek nem kezdődhetnek számmal vagy írásjellel.
- Nevek nem kezdődhetnek xml (vagy XML. vagy Xml. , stb) betűkkel.
- A nevek nem tartalmazhatnak szóközt.

XHTML

A HTML szabványt 1990 óta használják webes felületek megjelenítésére, weboldalak készítésére; eredetileg a dokumentum struktúrájának és tartalmának definiálására tervezték.

Bár a HTML rengeteget változott az első napok óta, sok része az első HTML dokumentációnak még mindig érvényben van a modern verzióban is, és az akkor definiált „HTML tagek" több mint fele most is létezik.

A HTML verziói		
Verzió	Melyik évben vált ajánlássá?	Jelenlegi státusz
HTML 2.0	1995. november	Elavult
HTML 3.2	1997. január	Elavult
HTML 4.0	1997. december	Elavult
HTML 4.01	1999. december	Elavult
HTML 5	2014 október	Ajánlás

HTML- nyelvről sokáig azt gondolták, hogy hamarosan felváltja az XHTML. Az XHTML az angol Extensible HyperText Markup Language kifejezés rövidítése. Magyarul: bővíthető hypertextes leírónyelv. A két nyelv között a különbség csekély, de a XHTML fő előnye az, hogy széles körben elfogadják a nem számítógépes eszközök, mint a mobiltelefok, PDA-k. Ezt eszközök közötti hordozhatóságnak nevezik.

XHTML szintén bővíthető, ami annyit jelent, hogy új tageket adhatunk hozzá, új dokumentumtípus deklarálása nélkül.

Az XHTML verziói		
Verzió	Utolsó módosítás	Jelenlegi státusz
XHTML 1.0	2000	Elavult
XHTML 1.1	2001	Nem ajánlják
XHTML 1.0	javítva 2002	Ajánlás
XHTML 2.0	2006	Nem vált ajánlássá

A HTML és az XHTML összehasonlítása:

HTML	XHTML
Az elemek és attribútumok nem érzékenyek a kis- és nagybetűkre, a <h1> ugyanaz, mint a <H1>.	Az elemek és attribútumok érzékenyek a nagybetűkre, mindent kisbetűvel kell írni.
Egyes elemekhez nem szükséges a bezáró tag (például a paragrafusoknál elég a), míg más esetekben (az ún. „üres elemeknél") nem szabad lezárni a taget (pl. a képeknél, <img>).	Minden elemet mindig le kell zárni (pl.: Egy paragrafus). A tartalom nélküli elemeket a kezdő tag végére írt perjellel is le lehet zárni (pl. a <hr></hr> helyett és a <hr /> -t kell használni).
Egyes attribútumok értékeit idézőjelek nélkül is meg lehet adni.	Az attribútumok értékeit mindig idézőjelekbe kell tenni.
Az attribútumokat bizonyos esetekben lerövidíthetők (pl. <option selected>).	Mindig meg kell adni a teljes attribútumot (pl. <option selected="selected">).

4.5.5. Tartalomhoz a forma: a stíluslapok szerepe

A CSS az angol Cascading Style Sheets kifejezés rövidítése, jelentése rangsorolt stíluslapok. A stíluslapot egy szöveges fájlban kell megírni, amit .css kiterjesztéssel kell elmenteni.

A HTML a tartalom szerkezetét, a CSS pedig a kinézetét írja le, de megfelelő tervezéssel precízen el is választható egymástól.

A megjelenés a betűtípusok, színek, margók, betűképek és egyéb stílus-vonatkozások, amiket a dokumentumban használnak. A Cascading Style Sheets (CSS) egy egyszerű stíluslap-szerkezetm amely lehetővé teszi a szerzőknek és olvasóknak, hogy stílust csatoljanak a weblapokhoz. Ez egy általános kiadványszerkesztés-terminológiát használ, amely megkönnyíti a szakértőknek és a gyakorlatlan tervezőknek a terméktulajdonságok kihasználását. Vizuális tervezési kérdéseket, mint például az elrendezést, így külön kell címezni a weblap logikai szerkezetében.

CSS egyik fő célja, hogy elválassza a dokumentum tartalmat a szerkezetétől A CSS-t arra használják, hogy a dokumentum tartalmához formázási, megjelenítési utasításokat kapcsoljanak, a HTML/XHTML/XML –t pedig arra, hogy a szerkezetét felépítsék. Ily módon, a tartalom és a forma kettéválik, ha egyszer a dokumentum szerkezetét helyesen alakítottuk ki, az oldal megjelenését egyszerűen módosíthatjuk, akár egész más dizájnt is tervezhetünk, anélkül, hogy a html állományainkhoz hozzá kelljen nyúlni.

A stíluslapban történt módosítás az összes olyan html oldalra hatással lesz, amihez hozzá van kapcsolva.

4.43. ábra Egységes formátum a különböző tartalmakhoz

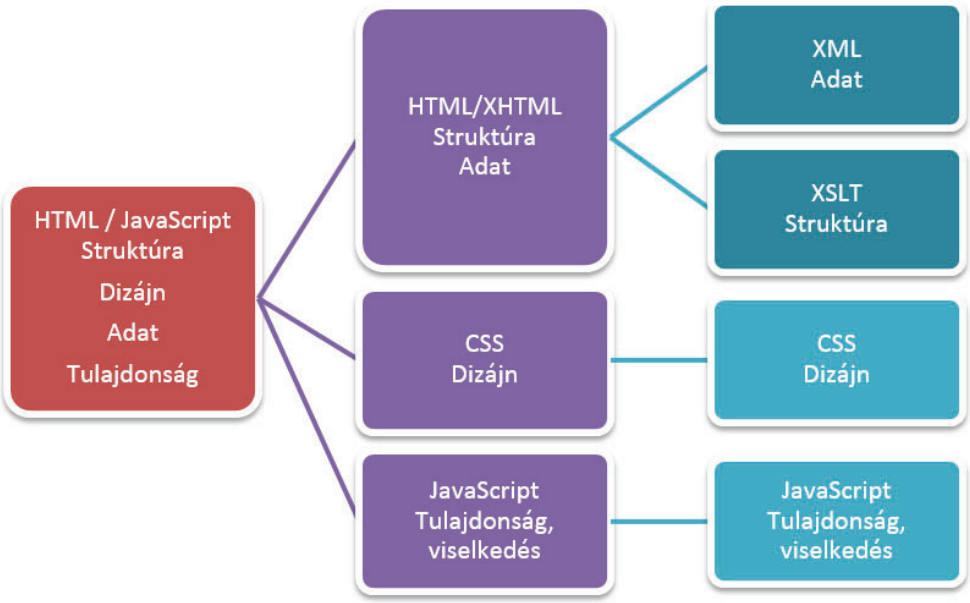
Érdemes ellátogatni az alábbi weboldalra, ahol megnézhetjük, hogy azonos tartalom mellett milyen különbözőképen jelenhet meg az oldal, csupán a CSS módosításával: http://www.csszengarden.com/

4.44. ábra Ugyanaz a weblap különböző megjelenéssel

Az XSL

Láttuk, hogy fontos célként jelenik meg a tartalom és a forma szétválasztása, ami a *css* segítségével lehetővé vált. Ugyanakkor a legtöbb dokumentum még tartalmaz a tartalomhoz tartozó strukturális kódot.

Az XSLT használata lehetővé teszi a tartalom teljes elválasztását a dizájntól. Ahelyett, hogy összekevernénk a strukturális jeleket az adatokkal, kettéválasztjuk a kettőt. A tartalom XML-ben található, a dokumentum struktúrája XSL-ben, a dizájn CSS-ben, a jellemzők pedig JavaScript/ECMAScript-ben. Az elkülönítés fejleszti a munkafolyamatokat. Ez a modell nemcsak a kezelhetőséget, de a nagyon összetett weboldalakat is fejleszti. Gyakorlatilag az XSL segítségével az XML nyelven írt adathalmazt html kódok segítségével átültethetjük XHTML formátumba, hogy egy böngészővel meg lehessen jeleníteni azokat.



4.45. ábra A tartalom és megjelenés szétválasztása

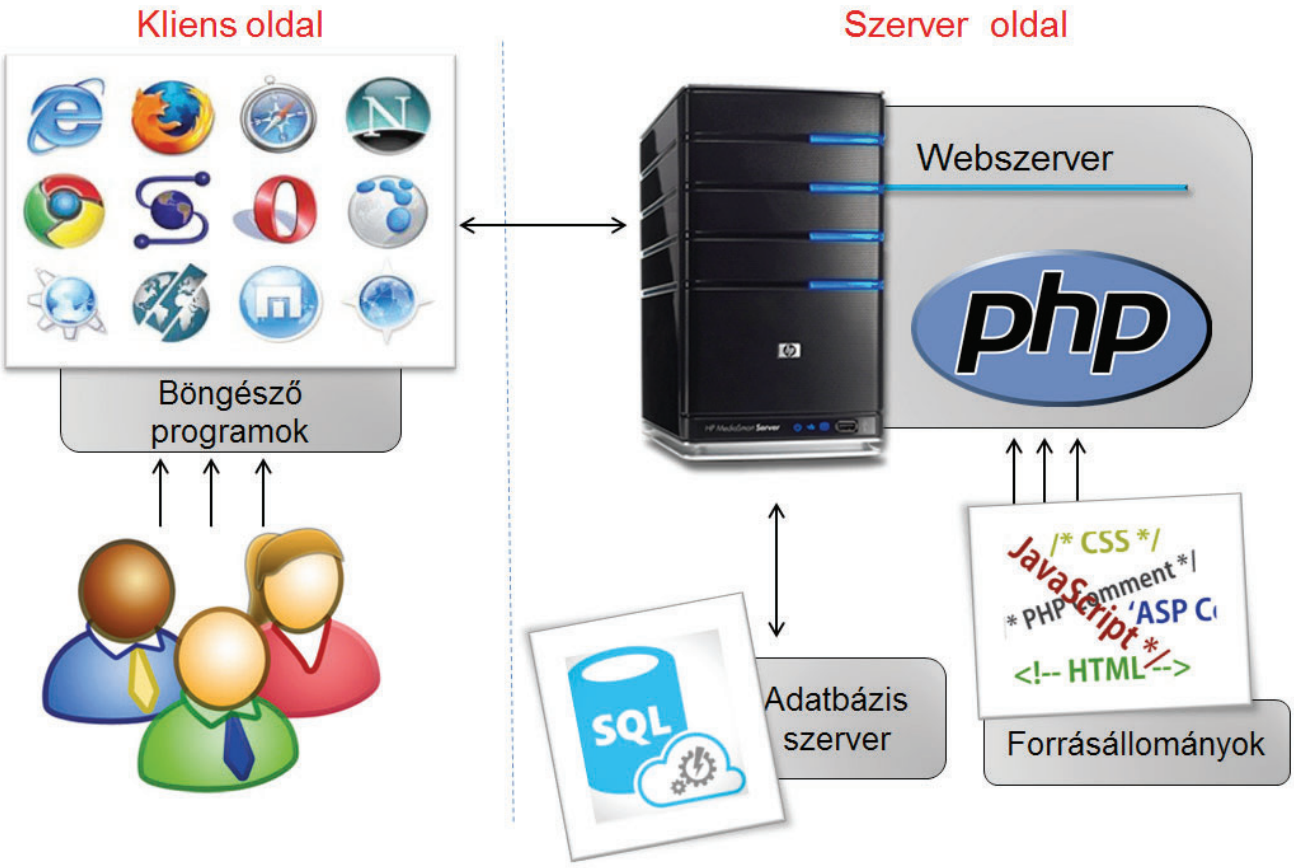
4.6. Webalapú programozás

Tanulási célok

- Különbséget tesz a kliensoldali és szerveroldali technológiák között, és felismeri a webalapú programozási nyelvek különböző típusait.
- Felsorolja egy webalapú rendszernek egy meglévő rendszerbe való integrálásakor fellépő legfontosabb kihívásokat.

4.6.1. A kliensoldali és szerveroldali technológiák

A világhálón elérhető dokumentumokat, weblapokat szerver gépeken tárolják. A weblapok elérése kliens-szerver architektúrában valósul meg. Amikor a böngészőbe beírjuk a weboldal URL címét, böngészőnk (a kliens) a http protokollon keresztül elküldi a kérést a célszerver felé, ahol a tartalom található. A szerver feldolgozza a kérést, majd a megfelelő feladatok elvégzése után visszaküldi a kívánt tartalmat a böngészőnek, amely megjeleníti azt.



4.46. ábra Szerver oldali feldolgozás

Kliens oldali programozás

A kliens oldali technológiák a web tervezésnél használt, a felhasználó számítógépén futó technológiákat írják le. Ezen technológiák használata esetén a böngészőnek nem csak a html kódok olvasása és megjelenítése a feladata, hanem pl. az oldalon elhelyezett Java Scriptek, Flash animációk futtatása is. A kliens oldali technológiák használata esetén szükség lehet a böngészőben egy speciális plugin használatára.

A Java Script egy kliens oldali szkriptnyelv, mely lehetővé teszi az Internetre a hagyományos asztali alkalmazásokhoz hasonlatos programok fejlesztését. Ezen programok végrehajtása a felhasználó böngészőjén történik, nem pedig a szerveren. Ezen programok írásához használt nyelvek közé tartozik a JavaScript és a Visual Basic Script.

Ezek a szkriptek lehetnek a HTML kódtól független, külön állományban, de be is ágyazhatjuk őket a HTML kódba. A felhasználó web böngészője végrehajtja a szkriptet, azután megjeleníti a dokumentumot, beleértve minden látható szkript kimenetet. Az utasítások a szerverrel való további kapcsolat nélkül is követhetők. A kliens oldali szkriptek nem igényelnek további kiegészítő szoftvert a szerveren, mindazonáltal elvárják, hogy a felhasználó web böngészője értse azt a szkriptelési nyelvet, amiben megírták őket.

Szerver oldali programozás

A CGI (Common Gateway Interface, általános átjáró felület) egy olyan felület, amely összeköti programunkat a HTTP szerverrel, ezen keresztül a felhasználóval. CGI programok írásához akármelyik programnyelv felhasználható, melyet az adott operációs rendszer támogat. Az így elkészített alkalmazásaink a WWW-n keresztül válnak elérhetővé.

A CGI program a **szerveren** kerül végrehajtásra még mielőtt az oldal továbbítódna a felhasználónak. Ilyen lehet például egy adatbázisból mondjuk egy PHP kóddal előállított adathalmaz, mely beépül az oldal tartalmába. A szerveroldali technológiák nem igényelnek a böngészőn kívül más kliensoldali erőforrást (pluginokat). Ellenben megkövetelik, hogy a nyelvi értelmező eszközük a szerveren telepítve legyenek, és ugyanaz a kimenet legyen a mindenhol, függetlenül az ügyfél böngészőjétől, az operációs rendszerétől, vagy más rendszerjellemzőktől.

4.47. ábra Szerver szoftver brandek logói

Szerveroldali szkriptek Perl, PHP és szerveroldali VBScript nyelven íródnak és a webkiszolgálók hajtják végre, amikor a felhasználó kéri a dokumentumot.

A kimenetek a webböngészők által érthető formátumban, általában HTML-ben jönnek létre, amelyeket szerver elküld a felhasználó számítógépére.

Biztonsági megszorítások miatt a kliensoldali szkripteknek nem engedik meg, hogy a böngésző alkalmazáson túl hozzáférjen a felhasználó számítógépéhez. Ha ki akarjuk kerülni ezt a megszorítást, egyéb technikákat kell használni az oldalon, pl. ActiveX Controls-t.

Összefoglalásként elmondhatjuk, hogy kliensoldali szkriptek jobban hozzáférnek a felhasználó böngészőjének információihoz és a funkcióikhoz, míg a szerveroldali szkriptek jobban hozzáférnek a kiszolgáló által elért információkhoz és funkciókhoz.

4.6.2. Webalapú rendszer integrálása

Webalapú rendszerek integrálásakor több különböző rendszer kombinálásával egy új, webböngészőn keresztül futtatható rendszert hozunk létre. Ez gyakran nem könnyű feladat.

Szembe kell néznünk néhány kihívással:

- Át kell gondolni, hogyan lehet átültetni a régebbi rendszerbe ágyazott üzleti szabályokat, hogyan lehet új munkafolyamatokat meghatározni és hogyan lehet az új rendszerbe való integrációt megvalósítani.
- Az eredeti alkalmazást valószínűleg nem platformfüggetlenül tervezték. Ebből kifolyólag több alkalmazás esetén össze kell hangolni a különböző szoftver és a hardver platformokat.
- Az új szoftverek teljesen különböző technológiákat használhatnak.
- Meg kell fontolni különböző üzleti szempontokat is, pl. mennyibe kerül az átállás, mekkora bevétel-növekedést fog eredményezni, a régi rendszer fenntartási költségei mikor haladják meg az új rendszerre való átállás költségeit stb.
- Az eddigi rendszert használó alkalmazottakat meg kell győzni arról, hogy támogassák az új rendszer használatát, illetve át kell képezni őket az új rendszer használatára. Ha pl. az új felület eltér az eddigi megszokott program felhasználói felületétől, megzavarhatja a felhasználót, és nehezíti az új program elfogadtatását az alkalmazottakkal.
- Egy új rendszer bevezetése szükségessé teszi a teljes biztonsági ellenőrzést. Amennyiben a programhoz tartozó adatbázis eddig lokálisan működött, most hogy lesz elérhető online? Van-e állandó Internetkapcsolat? Hány felhasználó tudja egyszerre használni az adatbázist? Bővíthető-e az szerver kapacitása? Az átállás során a régebbi adatokat is integrálni kell a rendszerbe az eredeti jogosultságok figyelembe vételével.

Miért is lehet szükség egy ilyen integrációra?

- Növelhetjük a kritikus üzleti folyamatok hatékonyságát.
- Csökkenthetjük a válaszidőt, gyorsítva a folyamatot.
- Növelhetjük az információ pontosságát.

4.6.3. Tesztkérdések

1.) Az alábbiak közül melyik NEM tartozik a kommunikáció résztvevői közé?

i. Küldő.
ii. Közeg.
iii. Üzenet.
iv. Vevő.

2.) Az alábbiak közül melyek a felhasználói felület hatékonyságát megjelölő mutatók?

A) Megtanulhatóság.
B) Megjegyezhetőség.
C) Hangalapú, vagy vizuális működés.
D) Elégedettség.

i. Csak A.
ii. Csak a C.
iii. A, B és D.
iv. Mindegyik.

3.) Mi jellemző a pixelgrafikus képekre?
A) A képek rácspontokból állnak.
B) Szabadon méretezhetőek minőségromlás nélkül.
C) Jellemzőjük a felbontás.
D) Matematikai modellekkel írjuk le a képet felépítő elemeket.
i. Csak A.
ii. A éc C.
iii. B és D.
iv. Mindegyik.

4.) Az alábbiak közül melyik tömörítetlen képformátum?
i. BMP.
ii. TIF.
iii. GIF.
iv. JPG.

5) Melyek igaz állítások az alábbiak közül?
A) A háttérszín-betűszín kombinációban a komplementer színek használata javasolt.
B) A színkörön szomszédos helyeket elfoglaló színek jól harmonizálnak egymással.
C) A képek elsődleges szerepe, hogy erősítse a szöveges tartalom információtartalmát.
D) Használjunk sok, nagyméretű, látványos képet az oldalunkon.
i. Csak A.
ii. B és C.
iii. A, B és D.
iv. Mindegyik.

6.) Az alábbiak közül melyik NEM jellemzi a hipertextet?
i. A tartalom lineáris felépítése.
ii. Az információk linkek segítségével érhetőek el.
iii. Az olvasók a tartalom böngészése során saját bejárási sorrendet alakíthatnak ki. .
iv. A hipermédia a hiperszöveg kiterjesztése, amely a szövegelemek mellett támogatja a kép-, hang- és videó elemeket is.

7.) Milyen navigációs lehetőségek állnak rendelkezésünkre egy weboldalon?
A) Navigációs oldalsáv.
B) Oldaltérkép.
C) Morzsamenü.
D) GYÍK.
i. A és C.
ii. B és D.
iii. A, B és C.
iv. Mindegyik.

8.) Mi határozza meg, hogy egy oldal felhasználóbarát-e?
A) Átlátható navigáció, könnyen megtalálhatóak a keresett tartalmak.
B) Sok és látványos kép használata.
C) Hosszú szöveges tartalom.
D) A legtöbb böngészőben helyesen jelenik meg és helyesen működik.
i. A és C.
ii. B és D.
iii. A és D.
iv. Mindegyik.

9.)Az alábbiak közül melyik állítás fedi le legjobban a jelölő nyelv fogalmát?

i. A jelölő nyelvek utasításai megmondják az értelmező programok számára, hogy az utasítások közé zárt szöveget miképp kell értelmezni, felépíteni, formázni, vagy feldolgozni, illetve mi az egyes szövegelemek szerepe.
ii. A jelölő nyelvek jellemzője, hogy az utasításokat és a tartalmakat szövegesen írja le, a kétféle funkció között nem tesz semmilyen módon különbséget.
iii. Jelölő nyelvek használatakor külön állományban tároljuk a tartalom megjelenési módját és külön állományban tároljuk a megjelenítendő tartalmat. A megjelenítendő tartalomba hivatkozásokat helyezünk el a másik állományban lévő formázási utasításokhoz.
iv. A jelölő nyelvek utasításai a tartalom strukturált megjelenítését határozzák meg. A nyelv elemei a tag-ek.

10.) Miből épül fel egy HTML tag?

A) címkék, vagy tag-ek.
B) attribútumok.
C) megjelenítendő tartalom.
D) attribútumok értékei.

i. A és C.
ii. B és D.
iii. A, B és D.
iv. Mindegyik.

11.) Az alábbiak közül melyek azok az elemek, mellyel NEM kell minden szabályos (validált) weboldalnak feltétlenül rendelkeznie?

i. Dokumentumtípus-meghatározás.
ii. <body> tag.
iii. <html> tag.
iv. <h1> tag.

12.) Az alábbi html kódok közül melyik helyes?

A	B	C	D
   kék sárga zöld 	 - kék - sárga - zöld 	Színek - kék - sárga - zöld 	 kék sárga zöld

i. Csak B.
ii. B és C.
iii. A és C.
iv. Csak az A.

13.) Az alábbi html kódok közül melyiket fogja elfogadni a validátor, ha a documentum típus meghatározása <!doctype html>?

A) <img src="images/kep.jpg" alt="képleírás" width="100" height="100" >
B) <img src="images/kep.jpg" alt="képleírás" width="100px" height="100px" >
C) <img src="images/kep.jpg" width="100" height="100" >
D) <img src="images/kep.jpg" alt="képleírás" >

i. Csak B.
ii. A és D.
iii. Csak a D.
iv. Csak az A.

14.) Az alábbi kódrészlet esetén mi igaz az alábbi meghatározások közül?

```
<table>
  <tr>
    <td>Vulkán neve</td>
    <td>Hely</td>
    <td>Utolsó nagyobb kitörés</td>
    <td>Kitörés típusa</td>
  </tr>
  <tr>
    <td>Mt. Lassen</td>
    <td>Kalifornia</td>
    <td>1914-17</td>
    <td>Explozív kitörés</td>
  </tr>
  <tr>
    <td>Mt. Hood</td>
    <td>Oregon</td>
    <td>1790</td>
    <td>Piroklaszt-torlóár és lávafolyás</td>
  </tr>
  <tr>
    <td>Mt .St. Helens</td>
    <td>Washington</td>
    <td>1980</td>
    <td>Explozív kitörés</td>
  </tr>
</table>
```

A) Az első sor második cellájának tartalma: „Hely”.
B) A 3. oszlop 3. sorába az 1790-es szám kerül.
C) A 3. sor 3 cellájába az Oregon szó kerül.
D) A táblázat hibásan fog megjelenni.

i. A és B.
ii. A és B és C.
iii. B és C.
iv. Mindegyik.

15.) Mi igaz az alábbi megállapítások közül az XML nyelvre?

i. AZ XML, a HTML nyelvhez hasonlóan, leíró nyelv.
ii. Az XML nyelvet az adatok megjelenítésére tervezték.
iii. A nyelvi elemek kötöttek, előre meg vannak határozva, akárcsak a HTML nyelvben.
iv. Attól függően, hogy milyen számítógépen fogjuk használni az elkészült kódunkat, különböző XML nyelvi elemeket kell használnunk.

16.) Mi a hiba az alábbi XML kódrészletben?

```
<?xml version=”1.0” encoding=”windows-1250”?>
  <konyv>
    <szerzo>Brian W. Aldiss</szerzo>
    <cim>Amíg világ a világ</cim>
  </konyv>
   <konyv>
    <szerzo>Isaac Asimov</szerzo>
    <cim>Alapítvány</cim>
  </konyv>
```

i. Csak a nyelv által előírt tag-eket használhatunk, ezért pl a <konyv></konyv> párosnak semmi értelme.
ii. Csak egyetlen gyökérelem lehet, itt pedig kettő is van.
iii. A tag-ek nem a magyar helyesírás szabályai szerint vannak írva (cím hosszú í-vel írandó!)
iv. Nem használhatunk windows-1250-es kódolást az XML nyelvben.