INFORMATIKAI RENDSZEREK TERVEZÉSE

Arisztotelész „Az egész több mint a részek összege”.

Már az i.e. 384-322-ig élő Arisztotelész is foglalkozott a rendszer jelenségével, viszont utána évszázadokig senki tette ezt értékelhetően. Egészen az 1900-as évekig kellett várni, hogy a figyelem középpontjába kerüljön, önálló diszciplinává pedig csak az 1950-es években vált a rendszerelmélet.

A **rendszer** különböző, de egymáshoz kapcsolódó elemek halmaza, amelyek együttműködve egy vagy több célt megvalósítanak. A rendszer működésére jellemző, hogy valamilyen bemenetből (input) egy folyamat végén kimenet (output) jön létre.

A rendszerek lehetnek egyszerűek vagy összetettek, **egyszerűek**, ha további alrendszerre nem bonthatóak, **összetettek**, vagy komplexek, ha felépítésük összetett, al- vagy részrendszerekre bontható. A rendszer lehet **nyílt** (a környezetükkel kölcsönhatásban áll) vagy **zárt** (nincs kapcsolat a külvilággal), elvont és konkrét, **determinisztikus** (egy adott bemenetre konkrét válaszokat ad, pl. számítógépes rendszerek), vagy **sztochasztikus** (egy adott bemenetből a folyamat végén többféle kimenetet eredményezhet).

Az **informatikai rendszer** az adatokat és információkat[1] tárolja, különböző rendszer specifikus módon feldolgozza, és elérhetővé teszi.

Az informatikai rendszerek nagyon sok félék lehetnek. Ennek reprezentálására nézünk meg néhány szempontot, amely alapján csoportosíthatjuk őket:

- Felhasználók száma szerint:
  - Egyfelhasználós
  - Többfelhasználós (multitasking)
- Hardver mérete alapján:
  - Mikro-

[1] Információ: az informatika alapfogalma. Az információ valamilyen tekintetben új hírt közöl, vagyis a tudásunkat gyarapítja. A modern, matematikai megalapozású elméletét Claude E. Shannon dolgozta ki az 1940-es és 50-es években.

    - Kis-
    - Nagygépes
- Processzorok száma alapján:
    - Egy processzoros
    - Több processzoros
- Rendszer architektúra alapján
    - Centralizált
    - Elosztott
    - Hálózati
- Feldolgozás időbelisége alapján:
    - Kötegelt
    - Interaktív
    - Valós idejű

Az informatikai rendszerek haszna a hagyományos információ feldolgozással szemben, hogy gyorsabb, precízebb eredményt adnak, a feldolgozható adatmennyiség óriási mértékben megnő, mindezek által költséghatékonyabb. Problémát jelenhet viszont, annak megoldása, hogy a rendszer általánosan használható legyen, ne csak egy esetben, továbbá a fejlesztés elhúzódásának megakadályozása, a karbantartás költségcsökkentése. Az informatikai rendszerek fontos kritériuma a megbízhatóság és biztonság, vagyis, hogy a rendelkezésre álljanak, az adatokat veszteségmentesen megőrizzék, fenntartsák az adatkonzisztenciát.

Az informatikai rendszer állhat kizárólag **szoftver** komponensekből, vagy **hardver** komponensekből, avagy a kettő ötvözetéből, sőt kihagyhatatlan, leggyengébb, de legkreatívabb eleme a rendszernek az ember (alkalmazók).

1. ábra A számítógépes rendszer rétegződése a tervezés során

A számítógépes rendszer elemei egymásra épülnek, sorrendjük nem cserélhető fel. Az elemek között **interfész**ek találhatóak, amelyek összekapcsolják őket, biztosítják a kommunikációt a két réteg között.

A hardver és a szoftver elemek tovább bonthatóak. A hardver architektúra elemei: központi egység (CPU), központi tár (Central Memory), sínek (bus), input/output perifériák (device).

A következőkben a számítógépes rendszerek tervezésével fogunk foglalkozni, de elsősorban a szoftverfejlesztés szemszögéből.

SZAKMAI INFORMÁCIÓTARTALOM

TERVEZÉSI ALAPISMERETEK

Komplex informatikai rendszereket, kizárólag tudatosan megtervezett munkamenet segítségével és annak következetes végrehajtásával lehet létrehozni. A tervezéstől és elemzéstől nem szabad az időt és energiát sajnálni, mert az utólagos módosítás még több időt igényel, valamint a követelményeknek nem teljesen eleget tevő és túl bonyolult rendszert eredményez.

Az informatikai rendszerek létrehozásának általános lépései:

- **Definíciós fázis:**
  - Cél meghatározás (a létrehozandó rendszer funkcióinak rögzítése a megrendelővel vagy a potenciális felhasználói igények felmérésével)
  - Erőforrások számbavétele
- **Fejlesztési fázis:**
  - Tervezés
  - Ellenőrzés, tesztelés (validáció)
  - Dokumentálás
- **Támogatási fázis:**
  - Átadás a megrendelőnek
  - Továbbfejlesztés, karbantartás (evolúció)

A következő tevékenységek a rendszer létrehozásának egészén végighúzódnak: projektmenedzsment, projektkövetés és ellenőrzés, dokumentációs tevékenység, kockázat elemzés és kezelés.

Az informatikai rendszerek ún. **életciklus modellek** mentén jönnek létre. Az életciklus magában foglalja mindazon tevékenységet, amely az informatikai rendszer követelményeinek meghatározásától, a rendszer létrejöttén, üzembe helyezésén és tartásán keresztül, egészen a rendszer használatának megszűnéséig tart. A modellek strukturálják a

tervezés és fejlesztés tevékenységét, útmutatást adnak a csoportmunka irányításához, meghatározzák a fejlesztésben résztvevők feladatait. A modellek abban térnek el, hogy a megvalósítás lépései nem azonosak, eltérő sorrendben, esetleg párhuzamosan hajtódnak végre, a különböző modellek más és más lépésekre helyezik a hangsúlyt. A következő fejezetben ezeket a modelleket fogjuk áttekinteni, de előtte még néhány dolgot tisztáznunk kell.

Közös a modellekben, hogy az informatikai rendszerek létrehozását, több lépésre bontják. A lépések, ha nem is épülnek közvetlen egymásra, mert például a fejlesztés során nem időben egymás után, hanem párhuzamosan végzik el őket, azonban minden egyes lépésnek összhangban kell lennie az összes többivel. Eme kritérium vizsgálatát jelenti a verifikáció. Amennyiben egy lépés nem tesz eleget a követelményeknek, úgy az nem hajtható végre, hanem módosítani kell, majd újra megvizsgálni.

A **verifikáció** azt vizsgálja, hogy az informatikai rendszer egyes elemei összhangban vannak-e a többi elemmel, együtt tudnak-e működni és, hogy a specifikációban megfogalmazottaknak eleget tesz-e. A verifikáció arra a kérdésre felel, hogy „jól működő terméket fejlesztünk-e?”.

A verifikációval nem összekeverendő fogalom a validáció.

A **validáció** az informatikai rendszer egészét vizsgálja, abból a szempontból, hogy rendszer megvalósítja-e a projekt kezdetekor meghatározott célokat, eleget tesz-e a rendszer a megrendelő igényeinek. Tehát a validáció arra a kérdésre válaszol, hogy „megfelelő terméket fejlesztettünk-e?”.

A teljes rendszer validálása viszont már túllép a célok ellenőrzésén és biztonsági feltételek teljesülését is vizsgálja.

Az informatikai rendszerek létrehozása sokszereplős. A résztvevőket két csoportra bontjuk: megrendelő/k vagy felhasználók és fejlesztők. A fejlesztőkön belül a feladatkörök szerint megkülönböztetjük a menedzsereket és az informatikusokat.

SZAKMAI INFORMÁCIÓTARTALOM

CÉL MEGHATÁROZÁS

Magától értetődően a létrehozandó informatikai rendszer céljának, követelményeinek, szolgáltatásainak meghatározásával kezdődik az informatikai rendszer megalkotása. A követelmények meghatározása több lépéses folyamat:

1. A követelményeket meghatározása
2. Megvalósíthatósági elemzés
3. A követelmények specifikálása és validálása
4. A követelmények dokumentálása

Két forrása lehet a követelményeknek: az elsődleges, ha egy megrendelő vagy megrendelő szervezet keres fel egy vállalatot, amely informatikai rendszerek tervezésével foglalkozik (**célszoftverek**). Avagy egy rendszertervező vállalat létrehoz egy szoftvert, amit később értékesít (**általános célú, dobozos szoftvertermékek**). Ez esetben a vállalat kijelöli a célcsoportot, és igyekszik meghatározni annak igényeit. Az az ideális, ha a megrendelő és a fejlesztő együtt dolgozik és közösen fogalmazzák meg a rendszer célját, a funkcióit, a teljesítményét, a felhasználói felületek milyenségét, a navigációs irányokat. Természetesen nem egyszeri tárgyalásról van szó, hanem megbeszélések és egyeztetések sorozatáról.

A **specifikáció** alapján azonosítja a fejlesztő a feladatot, és kezdi elemezni, hogy számára mit is jelent. Megállapítja az információ tartalmat, körülhatárolja a rendszer funkcióit, megpróbálja meghatározni, hogyan kellene működnie a rendszernek a kívánt cél eléréséhez, és rögzíti a felmerülő korlátokat is.

Nagyon gyakori, hogy a specifikáció véglegesítése előtt készítenek egy **modell**t, amely hozzásegít a rendszer várható működésének megértéséhez. Párhuzamosan a modellkészítéssel elvégezhető a **megvalósíthatósági elemzés**. A **megvalósíthatósági elemzés feladata** a követelmények megvalósíthatóságának, fenntarthatóságának a vizsgálata, annak érdekében, hogy a megrendelő és a fejlesztő helyes döntést hozhasson, a rendszer létrehozásának mikéntjéről (hiszen mindig többféle megvalósítási mód áll rendelkezésre).

Ezek alapján a megvalósíthatósági tanulmány a következőket tartalmazza:

- Projekt adatok
- A fejlesztés indításának okai és a célkitűzései
- A tanulmány alkalmazása során alkalmazott módszerek bemutatása
- Lehetséges alternatívák a fejlesztésre és azok elemzése
- Környezeti, gazdasági, társadalmi hatások vizsgálata
- Pénzügyi elemzés
- Kockázat elemzés
- A projekt megvalósíthatóságának vizsgálata

A megvalósíthatósági elemzés után a megrendelő és a fejlesztő újra összeül, és immár a végleges specifikációt állítják össze, ezt írásba is foglalják. Az így megszülető dokumentum a **követelmények dokumentációja** elnevezést kapja.

SZAKMAI INFORMÁCIÓTARTALOM

ERŐFORRÁSOK SZÁMBAVÉTELE

Az erőforrások számbavétele egyszerűbb feladat, mint a megrendelő igényeinek a felmérése.

Erőforrás alatt a projekt szempontjából releváns eszközöket, **hardver** elemeket, **szoftver**eket, **szaktudás**sal bíró embereket, és a rendelkezésre álló **adat**okat értjük.

Mivel nagyon sokféle lehet az a rendszer, amit létrehozunk, így nehéz lenne paradigmát adni arra, hogyan is tegyük az erőforrásuk biztosítását. Ráadásul nagyban pénzfüggő, aminek az elosztásával pedig a projektmenedzsment foglalkozik.

Magyarországon még nem, de külföldön egyre inkább gyakorlattá válik, már meglévő szoftverek, pontosabban **szoftver elemek újrafelhasználása (ún. komponens alapú programozás)**. Arra nincsen mód, hogy változtatás nélkül építsenek be az újonnan létrejövő szoftverbe már létező program elemet, de gondosan elkészített, sok paraméteres elemeket tudnak adaptálni a saját rendszerükhöz a programozók. Ezzel egyre inkább megszűnik a nulláról való fejlesztés gyakorlata. Ezzel a módszerrel idő, költség takarítható meg, és növelhető a rendszer biztonsága, hiszen az újrafelhasznált elem már jól tesztelt, gyakorlatilag hibátlan. Két hátrányos tulajdonsága van ennek a módszernek, az egyik, hogy egy újrafelhasználásra szánt szoftverkomponenst jóval bonyolultabb, drágább és időigényesebb megírni, a másik, hogy merevebbé válik a rendszer, és egy esetleges fejlesztés során az adaptált elem inkompatibilissé válhat.

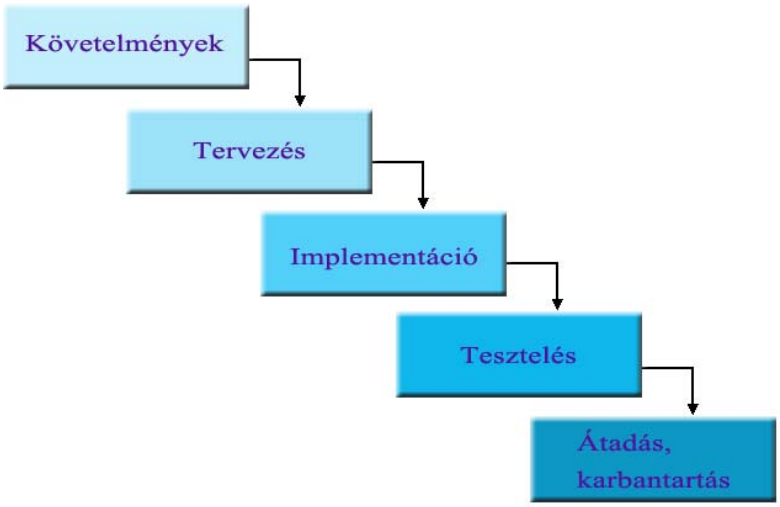
TERVEZÉS

TERVEZÉS

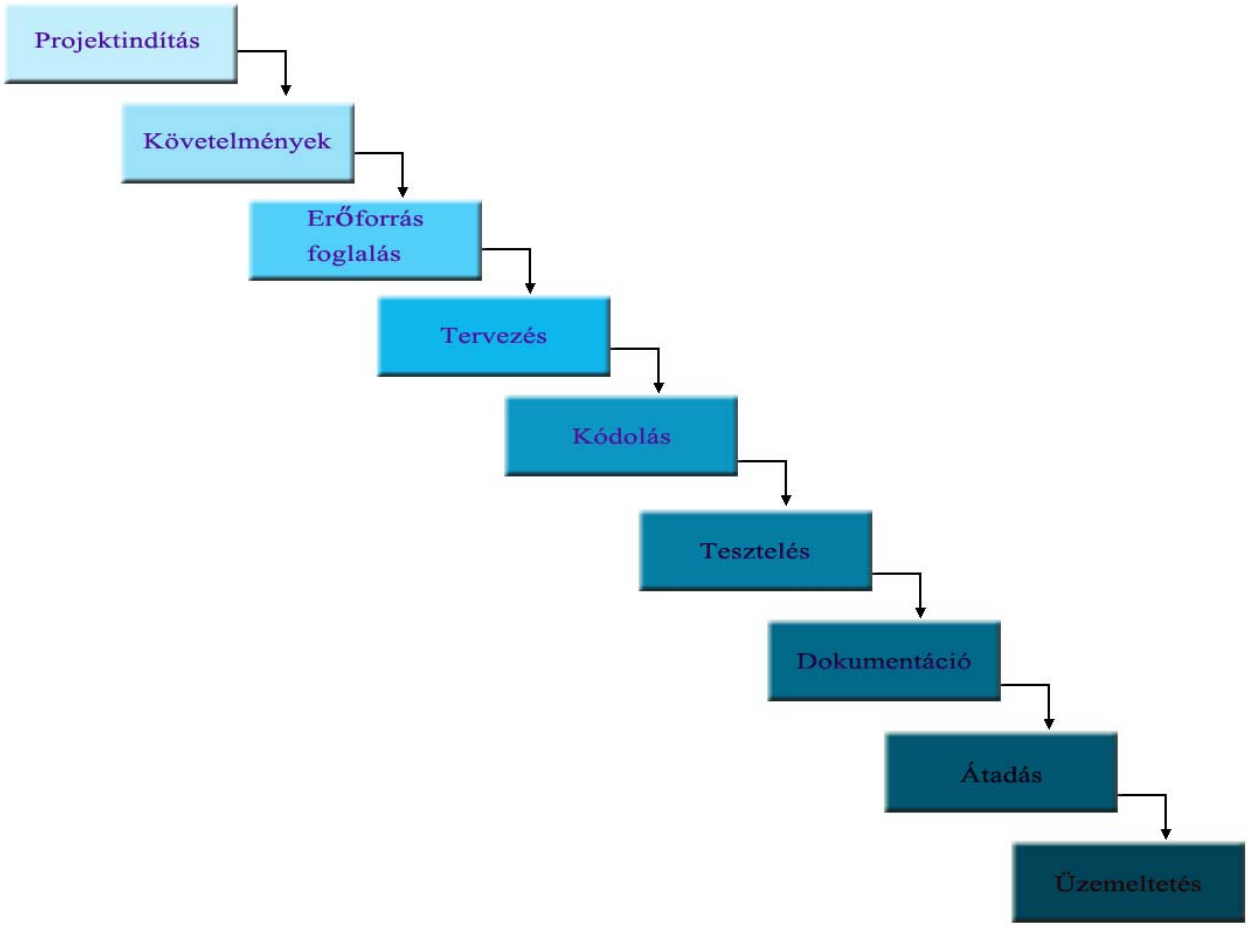
A tervezés lényege, hogy a specifikációnak megfelelő rendszert az adott körülmények között a leghatékonyabban megvalósítsuk. Mint mondtuk az informatikai rendszerek ún. életciklus modellek mentén jönnek létre. Először ezeket tekintjük át. Ugyanakkor a tervezés magában foglalja a rendszer architektúra tervezést, interfész tervezést, komponens tervezést, adatstruktúra tervezést, algoritmus tervezést is, ezért ezeket sem hagyjuk ki az áttekintésből.

Vízesés modell (waterfall)

1970-ben Winston W. Royce alkotta meg ezt a lineáris modellt, amelyben **a fejlesztési fázisok egymást követik** olyan módon, mint ahogyan a vízesésben lefelé zuhanva követik egymást a vízcseppek.



2. ábra Eredeti vízesés modell

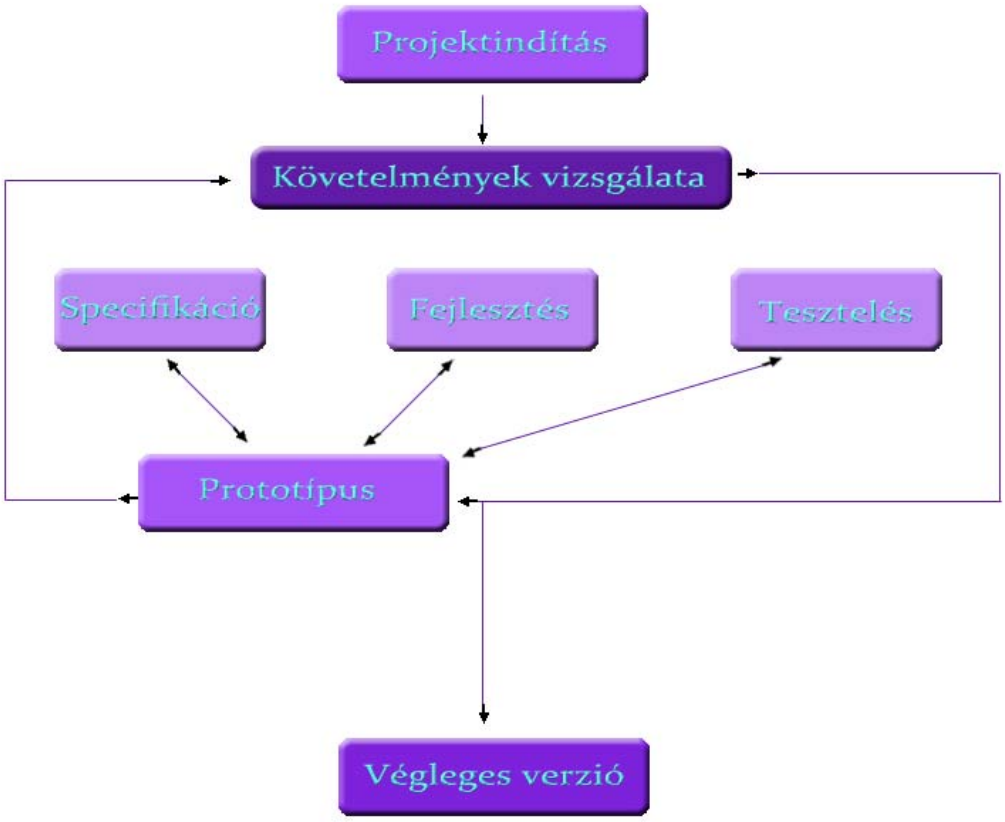


3. ábra Javított vagy továbbfejlesztett vízesés modell

Csakis az adott fázis befejezése után léphetünk a következőre, szépen lassan lépésről lépésre haladunk lefelé. A fázisok végrehajtási sorrendje kötött. Esetlegesen problémát jelenthet, hogy csekély a visszacsatolás. A rendszer az utolsó fázis befejezés után válik komplexszé, így ekkor hirtelen több hiba is felmerülhet, amelyek javítása nehézkes lehet. A vízesés modell a kötött követelményrendszerrel rendelkező, jól dokumentált, kis, egyfázisú projektek megvalósításához ideális.

Prototípus vagy evolúciós modell

Mint a neve is mutatja a modell lényege, hogy a rendszer tervezés elején létrehozunk egy rendszer prototípust, azaz egy egyszerű programot, amely a célokat elnagyolva megvalósítja. Ez a modell ellentétben az előzővel nem dokumentum központú és nem lineáris, hanem **gyakorlat orientált**. Az először létrehozott szoftvert fejlesztjük tovább, teszteljük, továbbépítjük, javítjuk az esetleges hibákat, hogy minél hamarabb létrejöjjön a célok mindegyikét megvalósító program. Míg a vízesés modellben egyszer zajlik le a célok meghatározása, a fejlesztés és a validáció, az evolúciós modellben minden új prototípus a célok pontosításával kezdődik, majd újabb fejlesztő és validáló szakasz következik. A célok, követelmények a rendszer tervezésének legelején nem kiforrottak. Ha esetleg idő előtt elfogy a pénz, akkor is jó eséllyel átadható a megrendelőnek egy nem befejezett, de használható termék. Ennek a modellnek a hátránya, hogy a fejlesztés nem átlátható, rosszul strukturált. Leginkább kis és közepes rendszerek fejlesztésére alkalmas.



4. ábra Prototípus modell

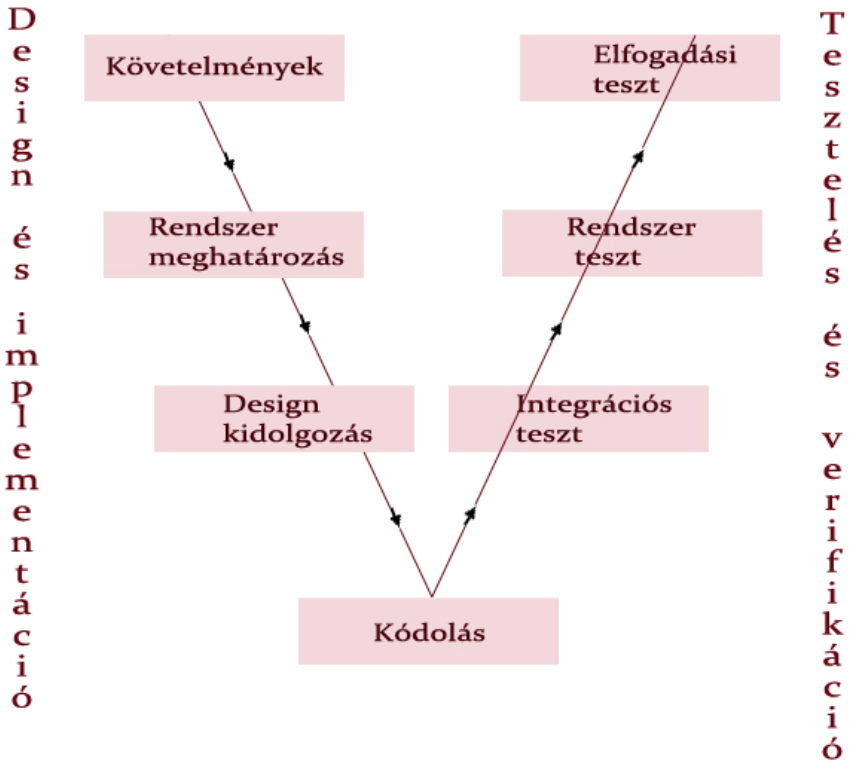
V modell

A vízesés modell továbbfejlesztett változata. Az 1980-as években a német védelmi minisztérium a következő szempontokat figyelembe véve alakította ki:

- A kockázat és költségek csökkentése

- Biztonság és a minőség biztosítása

Érdekesség, hogy tőlük függetlenül, de velük párhuzamosan USA-ban is kifejlesztették. 1992-ben a német hadsereg szoftverfejlesztési szabványává tette. **Tesztelés** és ennek következtében **biztonság közp**ontú. A V egyik szárán a tervezés, a másikon a tesztelés lépései állnak, vagyis a tervezés minden egyes elméleti lépéséhez tartozik egy számítógépen végzett tesztelési fázis. Tehát ebben az életciklus modellben egyensúlyba kerül a verifikáció és a validáció. A V modell előnye, hogy a megrendelő/ felhasználók jobban részt tudnak venni a munkában, mivel a fejlesztéssel egy időben zajló tesztelésben részt tudnak venni, és tapasztalataikat figyelembe vehetik a fejlesztők. Biztonságkritikus rendszerek[2] fejlesztéséhez kiváló.

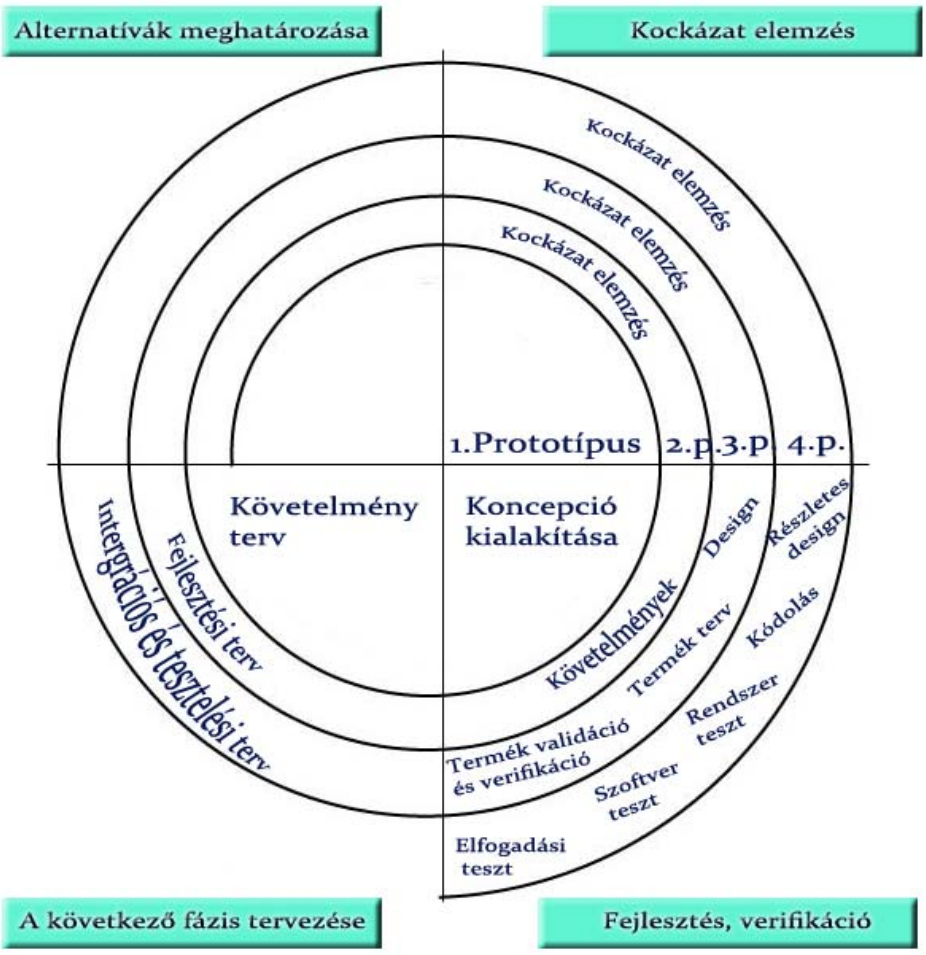


5. ábra V-modell

Spirális modell- Iteratív fejlesztés

[2] Olyan informatikai rendszer, amely üzemelésének legfőbb kritériuma, hogy ne veszélyeztessen emberi életet, egészséget, ne károsítsa a környezetet, és okozzon gazdasági károkat. Alkalmazási terület pl. egészségügy, vasúti közlekedés.

1986-ban Barry Boehm alkotta meg, a vízesés és a prototípus modell ötvözésével. A modell alapja az iteráció, amely folyamatosan ismétlődik a project során. A kiindulópont a spirál közepén a koncepció kialakítása (ez azonban a folyamat során változik), innen haladunk körkörösen „kifelé", előre a validálásig. Az iteráció lényege, hogy az egyszer már elvégzett munkafolyamatot később, folyamatos kockázat elemzés mellett, újra elvégezzük bővítve az előző eredményét. A tervezés során a kockázat elemzés mellett nagy hangsúlyt fektetnek a designra.



6. ábra Spirál modell

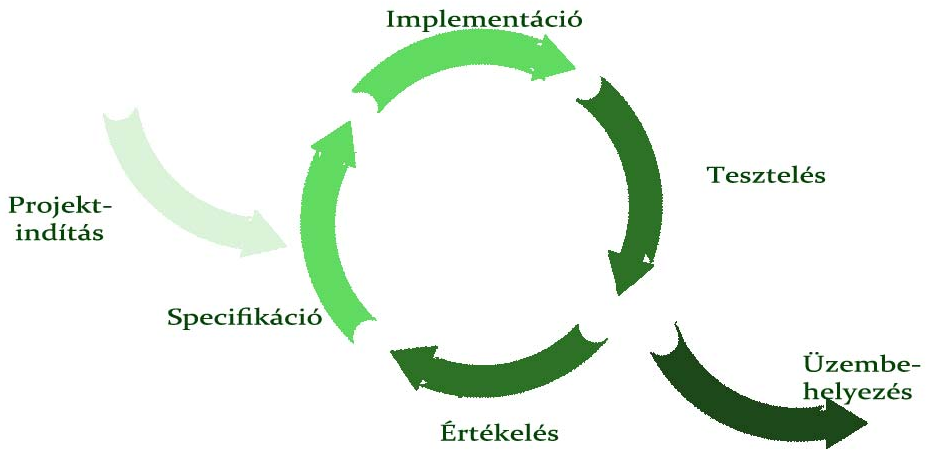
A kört négy negyedre oszthatjuk, de egy másik elképzelés szerint ez a következőképpen néz ki:

- Célok meghatározása, a koncepció kidolgozása,
- Kockázat elemzés,
- Fejlesztés, validáció és verifikáció,
- A project áttekintése, a következő iteráció tervezése.

Komplex koncepció, nehéz átlátni. Hasonlóan a V modellhez itt is könnyen bevonható a fejlesztésbe a megrendelő/felhasználó. Jól, sőt a végső célt tekintve túlzottan dokumentált fejlesztés (minden iteráció során elkészül a szükséges dokumentáció).

Iteratív- inkrementális modell

A spirális modellhez hasonlóan ennek a modellnek is az iteráció az alapja. Az előző modell annyival bővül, hogy a megvalósítandó célokat prioritásuk sorrendjében valósítja meg, és az iterációk végén az adott rendszer a követelmények egyre nagyobb hányadának felel meg. A fejlesztés során ún. inkremenseket hoznak létre. Az inkremens egy 20 000 ezer sornál kevesebb sort tartalmazó egység, amely legalább egy rendszerfunkciót megvalósít. A rendszer nem bonyolódik, hanem szélességében bővül. Ez annak köszönhető, hogy a legfontosabb funkciókat már a fejlesztés elején megvalósítjuk, az első inkremens képében, amit később a többi inkremenssel bővítünk. A megrendelő/felhasználó tehát az első inkremens létrejötte után használhatja a rendszert, a legfontosabb szolgáltatásokkal. A kockázat alacsony, hiszen már nagyon korán elkészül egy működőképes rendszer. A modell jól használható, ha a követelmények nem teljesen tisztázottak, hanem a projekt megvalósulásával párhuzamosan kristályosodnak ki.

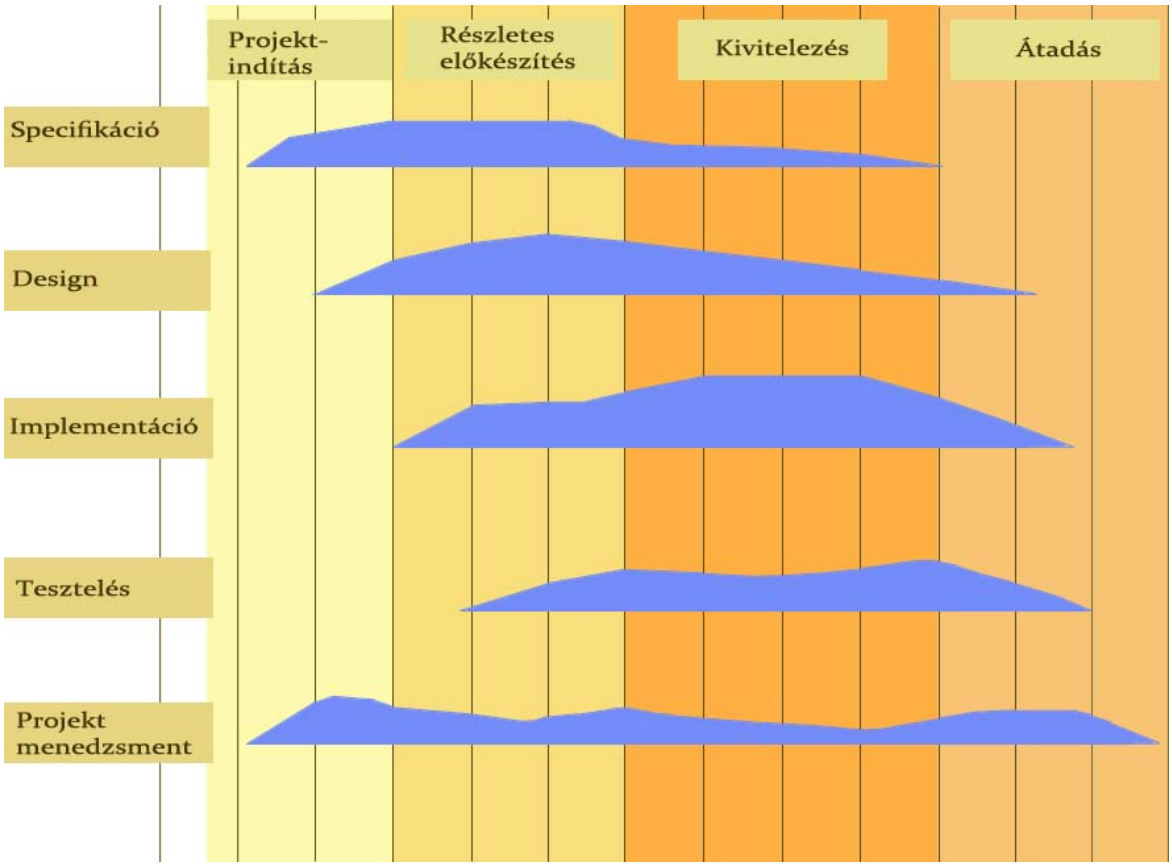


7. ábra **Iteratív- inkrementális modell**

Egyesített eljárás (Rational Unified Process)

1997-ben az összes addig létező tervezési modell pozitív tulajdonságait egyesítve alkották meg. Valójában egy keretrendszer, amelyet testre kell szabni. Jellemzői: **használati eset vezérelt** (a fejlesztés elején meghatározzák a használati eseteket, amelyek végigkísérik a fejlesztést), architektúra központú, iteratív és inkrementális.

Az ábra vízszintesen az időbeli haladást jelenti, függőlegesen a tevékenységeket látjuk. A sávok magassága a tevékenység intenzitását, erőforrás igényét mutatja. A fázisok véghezvitele során több munkafolyamat hajtódik végre. A különböző fázisokban eltérő a munkafolyamatok intenzitása. A fázisok száma kötött. Minden fázis végén található egy ún. **mérföldkő** (milestone), amely megvalósítása után lehet dönteni a további fejlesztésről, avagy annak befejezéséről. Egy eltérő szemlélet szerint a fázisokat iterációkra bontják. Az iterációk száma az adott projekttől függ, nem lehet a szabványban előre rögzíteni. Az iterációk végére egy működő rendszernek kell létrejönnie. Az iterációk lefolyása viszont kötött.



8. ábra Egyesített eljárás (Rational Unified Process)

Rendszerek működésének tervezése

Az életciklus modellekben ez a lépés a kódolás, szoftver architektúra kialakításának lépése. Két részre osztható:

1. Logikai rendszertervezés
2. Fizikai rendszertervezés

A logikai rendszerterv elemei:

- Funkcionális felépítés
- Felhasználói felület

- Navigációs struktúra
- Adatszerkezet terve

A fizikai rendszerterv elemei:

- Szoftver architektúra
- Alkalmazás fejlesztő eszközök meghatározása
- Szoftver erőforrás igénye
- Modulvázak
- Adat és objektum specifikációk

Az informatikai rendszerek alapja a hardver réteg, arra épül az operációs rendszer, amelyre pedig az alkalmazói programok. Feladatunk nagyon sokféle lehet. Előfordulhat, hogy a hardver architektúrát is nekünk kell kiépíteni, de az is lehet, hogy „csak" egy alkalmazói programot kell készítenünk. Emiatt a sokféleség miatt csak a logikai tervezéssel foglalkozunk, annak is az általánosan leírható részeivel.

A **program** az algoritmus és az adatszerkezet együtteséből jön létre.

Programozási módszerek:

- Strukturált
- Imperatív
- Deklaratív
- Objektum- orientált
- Függvény- orientált
- Szimbolikus

Az algoritmustervezéssel és készítéssel a programozás modul foglalkozik, az adatszerkezetekkel az adatbázis kezelés, de röviden a következő alpontban mi is kitekintünk rá. A következőkben a rendszer néhány komponensével foglalkozunk.

Felhasználói felületek:

Rendelkezhet egy rendszer akármilyen sok funkcióval, lehet nagy tárolókapacitású, gyors számítási sebességű, a legmodernebb technikával felszerelt, ha a felhasználói felület nem ergonomikus, esztétikus és egyértelmű, akkor az emberek nem szívesen fogják használni, és nem is tudják kiaknázni mindazt a szolgáltatást, amit nyújtani tudna.

A **felhasználói felület** (user interface) egy rendszer vagy program azon elemeinek összessége, amelyek az emberrel való kommunikációért felelősek, és lehetővé teszik az ember számára a rendszer irányítását.

Szoftvereknél a leggyakoribb felhasználói felülettípusok:

- **Parancssoros**: parancsbevitel billentyűzettel, a képernyőn is szöveges üzenetek láthatóak.
- **Szöveges**: átmenet a parancssoros és a grafikus felült között. A képernyőn szöveges alapú színes, vagy fekete-fehér felület esetén eltérő színárnyalatú szöveges cellák jelennek meg.
- **Grafikus**: szöveges és rajzos elemek együttese jelenik meg a képernyőn.

A felhasználói felület kialakításának szempontjai:

- Ergonómia: az ember számára a legkisebb erőkifejtést igénylő felület kialakítása a cél.
- Felhasználói jártasság felhasználása: a felhasználó előzetes ismereteinek felhasználása, ne akarjunk teljesen újat és egyben szokatlant alkotni.
- Egyértelműség: egyszerű letisztult ábrákat, rajzokat és rövid tömör kifejezéseket használjunk.
- Túlzsúfoltság kerülése: egyszerre ne kínáljuk fel az összes funkciót, rangsoroljuk őket, rendezzük őket osztályokba.
- Konzisztencia: hasonló műveletek, hasonló módon történő jelölése.
- Valóság visszatükrözése: pl. az adatrögzítő alkalmazás emlékeztessen a hagyományos papír alapú nyomtatványra.
- Visszacsatolás : az események előtt váljon egyértelművé a felhasználó számára, hogy hol van lehetőség műveletek végzésére, és hozzuk a tudomására, ha csinált valamit. A hosszasabb folyamatoknál jelezzük, éppen hogyan áll, kb. mennyi idő van még hátra a feladat végrehajtásból.
- Hibaminimalizálás: Nagyobb jelentőségű műveletek előtt kérjünk megerősítést, és tegyük lehetővé az egyes műveletek visszavonását. A hibákról küldjünk egyértelmű jelzést, lehetőleg feltüntetve a hiba okát.
- Olvashatóság: jóllehet a tipográfia külön szakma, de az olvashatóságot és az átlátható szövegelrendezést a minimum követelménynek tekinthetjük.
- Színek ésszerű használata: A színek segíthetik a tájékozódást, de zavarhatják is a felhasználót. A színhasználat,szubjektív ahogyan a felhasználói felület kialakítása is, de van néhány alapszabály. Korlátozott számú színt használjunk. Figyeljünk a színek közti harmóniára (legalább ne vibráljanak a színek egymás mellett), az olvashatóságra (sötét háttéren, ne sötét betűszínt használjunk). Az egyes funkciókat ne csak színekkel jelöljük, hanem valamilyen grafikai elemmel vagy szöveggel is, mert sok ember színtévesztő vagy színvak. De fordítva is igaz, hogy a szöveges üzenetekhez érdemes színt adni.
- Egér és billentyűzet használata: a legkönnyebb és gyorsabb kezelhetőség érdekében az egér mellett tegyük lehetővé a billentyűzet használatát is, pl. űrlapok esetében a szövegmezők közötti váltásra.

- Ablakok: érdemes a részfeladatokat elkülöníteni és külön ablakokat használni erre, de mértékkel, ne alakuljon ki kaotikus láncszerkezet. A körkörös hivatkozást mindenképpen ki kell küszöbölni.
- Egyéb: célcsoport sajátosságai, pl. kor vagy kulturális különbségek.
- Speciális igények: pl. gyengén látók.

Mindezekre akkor nyílik lehetőség, ha a felhasználói felületet szétválasztjuk a belső alkalmazáslogikától. Ráadásul így könnyen cserélhető, változtatható lesz a felhasználói felület.

A grafikus felület elemei:

- Fejléc
- Menüsor
- Munkaterület
- Egyéb sávok, pl.: keresősáv
- Nyomógombok
- Gördítő sáv
- Szerkeszthető és nem szerkeszthető szövegmezők
- Jelölőnégyzetek
- Legördülő listák
- Táblázatok
- Képek, ábrák etc.

Navigáció megtervezése, navigáció és interakciók fejlesztése:

A **navigáció** a mozgást jelenti a rendszerben, alkalmazásban.

Navigációra azért van szükség, mert az alkalmazások általában nem annyira egyszerűek, hogy minden szolgáltatásuk egyetlen képernyőfelületen elférjenek, hanem több „oldalra” van szükség. A szolgáltatásokat strukturálnunk kell, és logikusan rangsorolni őket. A navigáció kialakítása szorosan összefügg a felhasználói felület kialakításával. Hasonló alapelvek szerint alakítjuk ki, ráadásul gyakorlatilag egyszerre hozzuk létre őket. A tartalom logikus elrendezése azért fontos, hogy a felhasználó megtalálja, amit keres, és visszataláljon oda, ahol már egyszer járt, amit már egyszer megtalált (a weboldalak böngészése során könyvjelzőket használunk).

A navigációnak lehetővé kell tenni, hogy könnyen megtaláljuk, amit keresünk, tájékozódhassunk a rendszer szolgáltatásairól, és nyomon követhessük, hol járunk.

Honlapon bevett keresődobozok használata, viszont ez semmiképpen sem helyettesítheti a jó navigációt, csak kiegészítheti azt.

A navigációt megtervezhetjük hierarchikus szervezeti tábla segítségével, vagy diagramm, vagy egyéb folyamatábra segítségével.

Kommunikációs kapcsolatok (felületek) fejlesztése:

A kommunikációs felületeket ne keverjük a felhasználói felülettel!

A kommunikációs felület feladata, hogy kapcsolatot tartson a rendszer és a hardver eszközök között. Adatcserét és szinkronizációt valósít meg.

A teljes hardvereszközökre alapuló ipari komplexum irányítási rendszerétől, az egyszerű alkalmazásig bármit fejlesztési feladatként kaphatunk, mindegyik esetben más és más a kommunikációs felület szükséglete, és más kommunikációs csatornán keresztül kommunikálnak (pl. COM, CORBA)

A következő esetek állhatnak fenn a mi alkalmazásunk és a külső alkalmazások között:

1. **Passzív**: Alkalmazásunk küld kéréseket a külső alkalmazásnak,az kiszolgálja a rendszerünket, viszont a külső alkalmazás sosem küld kérést a mi alkalmazásunk felé. Alkalmazásunk nem eseményvezérelt
2. **Aktív**: A külső alkalmazás és a mi alkalmazásunk kölcsönösen kérésekkel fordul egymáshoz, és kiszolgálják egymást. A mi alkalmazásunk eseményvezérelt.

Adatszerkezetek, objektumok:

Az adatok tárolásában és feldolgozásában a számítógépek óriási áttörést jelentettek. Lehetővé vált nagy mennyiségű adat gyors és egyszerű feldolgozása. Ez annak köszönhető, hogy a számítógépen tárolt adatok **közvetlen elérésűek**.

Gondoljunk csak egy könyvtárra! Mindenki keresett már valamilyen könyvet életében, ehhez hajdanán (és még ma is sok helyen) vagy a könyvtáros memóriájára hagyatkoztunk, vagy kikerestettük vele, vagy mi magunk kikerestük a cédulakatalógusból a könyv céduláját. Ehhez oda kell sétálnunk a kétajtós szekrény méretű állványhoz és megkeresni az adott betűnek megfelelő rekeszt, majd azt kihúzni és sok minket nem érdeklő cédulát átnézve kiválasztani az áhított nyomtatványét. Ezzel szemben a számítógépes katalógusban egyetlen szó beírásával megtalálhatjuk, amit keresünk. Hogyan lehetséges ez? Az adatok a világ egy-egy valós objektumára vonatkoznak, jelen esetben egy könyvre.

Adatnak nevezzük a számokkal leírható információt, amely már korábban rögzítésre került.

Az adatokat felbonthatjuk, csoportosíthatjuk. Az adat legkisebb egysége az **adatelem**, amelyre külön hivatkozni is tudunk. Esetünkben az adatelem pl. a könyv címe vagy szerzője. Az adatelemeket egy adatszerkezet, a katalógus, foglalja magába.

Az **adatszerkezet** az adatelemek olyan véges halmaza, amelyben az adatelemek valamilyen módon összekapcsolódnak, és azonosítható szerkezetbe rendeződnek.

Az adathalmazok szerkezetét mi alakítjuk ki, annak érdekében, hogy rendezetté váljanak, és műveleteket (pl. keresés) végezhessünk rajtuk.

Az adatszerkezetek kialakítását általában három szempont befolyásolja:

1. Az adatok eredeti struktúrája
2. A rendelkezésünkre álló tárolóeszközök kapacitása, gyorsasága
3. Az adatfeldolgozási igények

A matematikai adatszerkezet egy <A, R> rendezett pár, ahol az A: adatelemek halmaza, R: a halmazon értelmezett valamilyen reláció.

Adatszerkezetet szerkezet szerinti csoportosítása:

Homogén adatszerkezetek: Az azonos típusú adatelemekből állnak.

1. Struktúra nélküli adatszerkezet
    a. Az adatelemek között nincsen kapcsolat
    b. Pl.: halmaz
2. Szekvenciális adatszerkezet
    a. Az adatelemek egymás után helyezkednek el. Az adatok között egy-egy jellegű a kapcsolat. Minden adatelem csak egy helyről érhető el.
    b. Pl.: egyszerű lista.
3. Asszociatív adatszerkezet
    a. Az adatok valamilyen közös tulajdonság alapján kerülnek bele. Résztulajdonságok alapján részhalmazokat hozhatunk létre belőle. Az adatelemek között nincs kapcsolat. Az elemek egyedileg címezhetőek.
    b. Pl.: tömb, ritka mátrix, tábla
4. Hierarchikus adatszerkezet
    a. Az adatelemek között alá-fölérendeltségi viszony van. A kiindulópont a gyökér, amelyből kiindulva az összes többi elem elérhető. Az adatelemek közti kapcsolat egy : sok jellegű.
    b. Pl.: fa, összetett lista
5. Hálós adatszerkezet
    a. Nincs kitüntetett elem, elvileg minden elemtől több irányba is el lehet indulni, és egy adatelemet több irányból elindulva el lehet érni. Az adatelemek között sok: sok kapcsolat van.

Heterogén adatszerkezetek: különböző típusú adatelemekből épülnek fel.

Adatszerkezetek memóriában történő helyfoglalása szerinti csoportosítása:

Statikus adatszerkezet: Előre meghatározott számú elemből épül fel. Az elemek száma nem növekedhet, csak az egyes elemek mérete. Pl.: tömb, rekord, halmaz

Dinamikus adatszerkezet: Az adatelemek száma változhat, nem meghatározott. Pl. Lista, gráf, fa.

A leggyakoribb adatszerkezetek:

Verem: angolul stack.

Homogén adatszerkezet, amelyen két művelet értelmezett: új elem elhelyezése a verem tetején (push), új elem kivétele a verem tetejéről (pop). A legutoljára berakott elemet tudjuk legelőször kivenni. **LIFO**-nak is szokták nevezni az angol **Last In First Out kifejezés után**.

A verem önmagában nem létezik, hanem nekünk kell megvalósítani. Kell hozzá:

1. Egy vektor az elemek tárolásához
2. Két mutató, az egyik a verem alját, a másik a verem tetejét mutatja

Határozzuk meg a verem adatszerkezetét:

```
Konstans Maxelem=maximális_elemszám
    típus Elemtípus: tárolandó_elem_típusa
    típus Veremtípus: rekord(
            Elem: tömb [1…maxelem] Elemtípus
    Alja, Teteje: egész
)
```

A verem kezelése:

Kezdőállapot beállítása, a verem inicializálása:

Alapállapotban a verem üres, viszont ez nem magától értetődő, be kell állítani:

```
eljárás VeremKezd(Verem:VeremTípus)
        Veremalja:=1
        Veremteteje:=0
vége
```

Értéket a := jellel tudunk adni. az eljárást meg kell kezdeni, és be kell fejezni.

Fontos, hogy megtudjuk a verem tele van-e vagy üres-e, hiszen csak ennek függvényében tudunk beletenni, vagy kivenni elemet:

```
függvény VeremÜres(Verem: VeremTípus): logikai
VeremÜres:= (Veremteteje<Veremalja)
vége
```

A verem akkor üres, ha a teteje az alja alá mutat. Ekkor a Veremüresf függvény igaz értéket ad.

```
függvény VeremTele(Verem: VeremTípus): logikai
VeremTele:= (Veremteteje:=MaxElem)
vége
```

A verem akkor van tele, ha a verem teteje mutató elérte a tömbhatárt, ekkor a VeremTele függvény igaz értéket ad.

Új adatelem elhelyezése a verembe:

```
függvény Verembe(Verem: VeremTípus; Adat: Elemtípus): logikai
    ha nem Veremtele(Verem) akkor
        Veremteteje:=Veremteteje+1
        Veremelem(Veremteteje):=Adat
        Verembe:=igaz
        különben
        Verembe:=hamis
    hvége

vége
```

A verembe csak akkor tudunk új elemet tenni, ha nincs tele. Ezért egy ha feltételt alkalmazunk, ha tele van a verem, akkor hamis értéket kapunk, ha nincs tele, akkor továbbhaladunk. A verem tetejére kerül az új elem, ezért a veremmutató, a veremtetejét eggyel feljebb kell állítani.

Adatelem kivétele a veremből:

```
függvény Veremből(Verem: VeremTípus; Adat: Elemtípus): logikai
    ha nem Veremüres(Verem) akkor
        Adat:=Veremelem (veremteteje)
        Veremteteje:=Veremteteje-1
        Veremből:=igaz
        különben
        Veremből:=hamis
    hvége
vége
```

Csak akkor tudunk elemet kivenni a veremből, ha van benne elem, vagyis nem üres. Így azt kell megvizsgálnunk először. Ha nem üres, akkor az Adat változóba beletesszük azt az elemet, amire a veremteteje mutat. Mivel a verem tetejéről kivettünk egy elemet, ezért a veremteteje mutatót eggyel lejjebb kell állítani.

Sor (Queue): Homogén adatszerkezet. Két alapművelet értelmezhető rajta:

1- Új elem elhelyezése sor végén (put).
2- Elem kivétele a sor elejéről (get).

A sor elnevezés nagyon találó erre az adatszerkezetre, mert pont úgy működik, mint egy sor, azaz, aki először áll be a sorba, az kerül először a kívánt helyre (pl. a pénztárhoz). **FIFO**-nak is nevezzük az angol **Firts In First Out** kifejezésből.

Ciklikus sor: Az egyszerű sortól annyiban különbözik, hogy nem csak a sor végére helyezhetünk elemeket, hanem, ha a sor elejéről vettünk már ki adatelemet, akkor az ott felszabadult helyre is tehetünk be új elemet.

Az eddigi adatszerkezetek hátránya:

Adatelemeket csak úgy lehet törölni vagy beszúrni, ha az utána álló összes elemet elmozgatjuk.

A tömb méretét előre meg kell határozni.

Láncolt lista: szekvenciális adatszerkezet. Mindig csak annyi memóriát használ, amennyire valóban szüksége van. Így viszont a memóriában az elemek nem egymás mellett helyezkednek el, hiszen pl. ha be akarunk rakni egy elemet, akkor keres a memóriában egy üres helyet, de az nem valószínű, hogy az utolsó elem mellett lesz. Azért, hogy az elemek ne vesszenek el, „össze kell láncolni" őket. Tehát a lista mindegyik eleme az adatelemen kívül arra nézve is tartalmaz információt, hogy melyik a következő elem.

Fa: az adatokat hierarchikusan tárolja. Ahogyan a faágak a törzsből ágaznak ki, úgy ebben az adatszerkezetben is egy kiindulópontból ágazik szét a szerkezet. Minden elem egy másik elemből származik, de belőle több elem is származhat. Ahogyan az embernek is csak egy édesanyja lehet, de egy édesanyának több gyermeke is lehet. Jegyezzük meg, hogy az adatelemeket csomópontoknak nevezzük.

Bináris fa: Olyan fa, amelyben a csomópontoknak pontosan két leszármazottja van.

Keresőfa: Olyan bináris fa, amelynek minden csomópontjára igaz, hogy a benne tárolt érték nagyobb, mint a bal oldali részfájában tárolt bármely érték, és kisebb, mint a jobboldali részfájában tárolt érték. Mint neve is mutatja kereséseknél használják leginkább, mert gyorsan lehet benne adatot keresni.

Gráf: heterogén adatszerkezet, tehát többféle típusú adatelemből épül fel.

Eseménykezelés

Az alkalmazáslogika megvalósítása során határozzuk meg az eseménykezelés mikéntjét.

Feladata a felhasználói felületről és a kommunikációs felületről érkező üzenetek és utasítások kezelése.

Az esemény valamilyen történés, amire a rendszernek reagálnia kell, pl. kattintás az egérrel. Az esemény bekövetkeztekor egy jelzés generálódik, ahová a jelzés eljut, ott kezelődik az esemény. A kezelés megváltoztatja az addigi futását a rendszernek. Az eseményt generálhatja a rendszer vagy a felhasználó is.

Az eseménykezelés teljesen programozási nyelvfüggő. Javában az események is objektumok. 4GL környezetben előre definiálható a felhasználói és a kommunikációs felületekhez kapcsolódó események köre. Flashben minden azt eseménynek nevezzük, ami a mozi futása közben történik. Az eseményeket objektumok figyelik és reagálnak rá. Belátható, hogy részletes ismertetése nem ennek a fejezetnek a feladata.

Állománykezelés

Állománynak nevezzük logikailag összefüggő egy egységbe tartozó adatok halmazát.

Az állománynév azonosítja az állományt, a típusát pedig a kiterjesztése jelöli. Állománykezelésnek hívjuk az állományok manipulációját. A kezelendő állomány típusa, elérési útvonala, mérete, típusa, a végrehajtandó feladat is teljesen eltérő, mindezen túl, hasonlóan az eseménykezeléshez rendszer és programfüggő a megvalósítása.

SZAKMAI INFORMÁCIÓTARTALOM

ELLENŐRZÉS, TESZTELÉS (VALIDÁCIÓ)

Korábban szó esett már a verifikációról és a validációról ez azonban még nem jelenti azt, hogy tökéletes programot hoztunk létre. ezért szükség van **tesztelésre, aminek a célja a szoftver hibáinak kiszűrése**.

A **tesztelés** kontrolált körülmények között zajlik, gyakorlatilag a rendszer ellenőrzött futtatása, és az eredményül kapott adatok kiértékelése.

Történhet részegységenként, vagy kiterjedhet az egész szoftverre. Az egyik nem teszi feleslegessé a másikat. Belátható, hogy egy kisebb részegységben, azaz komponensben egyszerűbb kiszűrni a hibákat, de előfordulhat, hogy a komponensek önmagukban helyesen működnek, de a komplex rendszer már nem. A verifikáció éppen ennek a megoldására irányul. A következő csoportosításban folynak a tesztek: egység, modul, alrendszer, rendszer. A modul egységekből áll, az alrendszer modulokból, a rendszer az összes alrendszert összefogja. A tesztelést két irányból érdemes lefolytatni a hierarchiában, fentről lefelé, és lentről felfelé is. A tesztelés történhet a program futatásával, vagy anélkül. Az előbbit **dinamikus**, utóbbit **statikus** tesztelésnek nevezzük. A program futtatás nélküli tesztelése a kód, dokumentáció „átolvasását" jelenti (többféle technika ismert: program vizsgálat, matematikai vizsgálat, statikus program analizátor, tisztaszoba technika). Ez a módszer azonban nem tudja felderíteni, hogy a program helyesen működik-e.

A tesztelésnek nevezzük a rendszer terhelhetőségének, teljesítményének a vizsgálatát is. A tesztelés általában a következő lépésekből áll: a megrendelő igényeinek a megismerése, tesztelési terv készítése, teszt szkriptek, más néven **teszt esetek** tervezése, teszt adatok tervezése, futtatás teszt adatokkal, teszt eredmény értékelése.

Jóllehet a pénzügyek a projektmenedzsment témakörébe tartoznak, de mégis érdemesnek tartjuk megemlíteni, hogy a tesztelés költsége a szoftver fejlesztésének előrehaladásával párhuzamosan növekszik. A követelmények meghatározásnak a szintjén a legegyszerűbb kiszűrni a hibákat, később a rendszer komplexebbé válásával bonyolultabb és költségesebb.

A hibáknak több szintje vagy típusa lehet:

- Bemeneti/kimeneti hibák: helytelen bemenet elfogadása, helyes bemenet el nem fogadása, hibás kimenet, helyes de hiányos kimenet
- Számítási hiba: helytelen algoritmus
- Interfész hibák
- Logikai hibák: hiányzó esetek, redundáns esetek, hiányzó feltételek,
- Adathiba: helytelen formátum, inkonszisztens adat.

Teszteket a program vagy rendszer fejlesztésének több szakaszában végzik, és ezek eltérő célokat szolgálnak, ez alapján a következő teszt típusokat különböztetjük meg:

- **belövési tesztek (debuggolás):** a debuggolás hibakövetést jelent. A program belső működését figyeljük meg töréspontok, vagy lépésenkénti végrehajtás mentén. Nem csak a hiba meglétét, hanem a helyét is megmutatja.
- **Részegység teszt**: egy rendszer vagy szoftver komponens önmagában, a rendszer többi elemétől független vizsgálata.
- **Egységtesztek és integrációs tesztek**: Mint mondtuk a részegységek önálló megfelelő működése nem elegendő, az egész rendszernek is jól kell működnie. A legcélravezetőbb módszer a komponensenkénti, vagyis inkrementális tesztelés. Az inkremenseket egyenként fűzzük össze és minden egyes hozzáfűzés után tesztelünk.

- **Stressz**teszt: a rendszer egészének működését teszteli. Célja a rendszer teljesítményének felmérése szélsőséges körülmények között is. Két funkciója van, megvédeni a rendszert a túrterhelés káros következményeitől (pl. adatvesztés), és a szélsőséges körülmények között kibukhatnak olyan hibák, amelyek normál körülmények között csak nagyon lassan.
- **Elfogadási teszt (acceptance teszt)**: az üzembe helyezés előtti utolsó teszt, amely már valós adatokkal zajlik. Ezt alfa tesztelésnek nevezzük. Ilyenkor dokumentálják, hogy a program eleget tesz-e a megrendelő elvárásainak.

Ahhoz, hogy a lehető legtöbb hibát kiszűrjük, nem elég random módon tesztelni a rendszerünket, hanem szükséges szisztematikusan tesztelni, ezért tesztelési tervet kell készíteni. A tesztelési terv alapja a rendszer követelményjegyzéke. Azt azonban ne feledjük, hogy a programot teljes mértékben lehetetlen letesztelni, minden hibát kiszűrni. Nézzük sorban a tesztelési módszereket!

Funkcionális tesztelés/ Fekete doboz módszer (Black-box testing): Az alapelképzelése, hogy a program működése nem ismert, azaz fekete doboz, csak az a tudvalevő, amit „belerakunk" és „kiveszünk". A cél az, hogy olyan bemenetet adjunk, amivel hibás kimenet jön létre. A teszt során felhasználjuk a specifikációkat, vagy a program egy korábbi változatát is. A bemeneteket valamilyen módon osztályozzák, hiszen véletlen bemenet adásokkal kicsi az esély a teszt sikerességére. A teszt előnye, hogy független a program implementációjától, vagyis az implementáció fejlesztésével együtt történhet a teszteset fejlesztés. Hátránya, hogy csak a hiba voltára derül fény, a helyére nem. A fekete doboz módszernek több altípusa van: határérték vizsgálat, ok- hatás analízis (döntési tábla alapú vizsgálat), ekvivalencia-osztály vizsgálat.

Strukturális tesztelés (white- box testing): Az előző teszttel ellentétben itt éppen a működési logika alapján tesztelünk. Nagyon sok tesztet igényel, mert a cél, hogy a program minden utasítása legalább egyszer végre legyen hajtva, vagyis minden lehetséges útvonalat végigjárassunk a programmal. A teszt előnye, hogy a hiba lokalizálható, és ez által könnyebben javítható. Viszont a hiányos vagy hibás szoftverspecifikáció nem tárható fel.

Szürke doboz tesztelés (Gray- boksz testing): Egyesíti a fekete és a fehér doboz tesztet. Némileg hasonló a verifikáláshoz, mert a program komponensek együttműködését is vizsgálja. A program működésének bizonyos részét ismertnek tekintjük, más részét nem. A hiba helye ezzel a módszerrel szintén meghatározható.

Útvonal- tesztelés: A fehér-doboz módszerhez hasonló, mert itt is az a cél, hogy minden végrehajtási útvonalat bejárassunk a programmal. Ezt a vezérlési folyam gráf segítségével végezzük, akkor tekinthető egy útvonal újnak, ha olyan csomópontot érint, amit az előző útvonalak, még nem érintettek. Minden feltételes utasítást letesztelünk igaz és hamis esetre is.

Objektumorientált tesztelés: A fehér doboz tesztelés kiterjesztésével jött létre. Külön teszteljük az objektumokat (a rájuk vonatkozó összes műveletet és attribútumot is), objektumcsoportokat és osztályokat is az összes bejárható útvonal mellett.

A teszteket általában a fejlesztő cég programozói végzik, de sokszor előfordul, hogy ún. béta verziót jelentettnek meg, amelyet a leendő felhasználók használatba vesznek, és jelzik a problémákat a fejlesztőknek. Ezzel ellentéte elképzelés a tesztek automatizálása, azaz teszt eszközök létrehozása. A teszt eszközök általában keretrendszerek, amelyeket saját programunkhoz kell beállítani.

SZAKMAI INFORMÁCIÓTARTALOM

DOKUMENTÁLÁS

Dokumentálás részben a fejlesztéssel párhuzamosan, részben a fejlesztés és tesztelés befejeztével zajlik. Célja a létrejövő rendszer funkcióinak, megalkotásának leírása, felhasználási feltételeinek, módjának megadása.

Fejlesztési dokumentáció:

Követelmények dokumentációja : Az első megszülető dokumentum a tervezés során. Ebben rögzítik, hogy az elkészítendő informatikai rendszer milyen szolgáltatásokat fog nyújtani, milyen követelményeknek kell megfelelnie. Tulajdonképpen a rendszer specifikációja.

Projektterv: A projekt minőségétől függetlenül tartalmazni kell egy erőforrás- és időszükségleti tervet. Az erőforrástervben meghatározzuk a létrehozandó informatikai rendszerhez hány és milyen végzettségű ember, milyen szoftverek és hardver eszközök szükségesek. Az időszükségleti terv az informatikai rendszer létrehozásának lépéseihez megmondja mennyi idő szükséges (vagy áll rendelkezésre) a kivitelezéshez. A projektterv részletesen leírja az informatikai rendszer megvalósulásának lépéseit.

Forráskód: egy konkrét programozási nyelven a célul kitűzött informatikai rendszer algoritmusa.

Programozói kézikönyv: helyettesítheti a forráskódot, lehetővé teszi, hogy egy képzett programozó megértse a szoftver működését, képes legyen az módosítani és tesztelni.

Tesztelési terv és eredményei: a rendszer vagy szoftver tesztelési koncepciójának, a teszteseteknek és a tesztelés eredményeinek a leírása.

Kereskedelmi dokumentáció:

Licence szerződés (licence agreement): az informatikai rendszer fejlesztője/ fejlesztő szervezete és a felhasználó között létrejött megállapodás, amely tartalmazza a rendszer felhasználásának módját, körülményeit, időtartamát, feltételeit, a két félre vonatkozó kötelezettségeket. Rögzíti, hogy a felhasználó milyen jogosultsággal használhatja a terméket, másolhatja-e (kereskedelmi szoftverek esetében általában egy biztonsági másolat készítése megengedett), tovább adhatja-e. Ezek a szerződések blanketta-típusúak, vagyis a felhasználó nem módosíthatja őket, vagy elfogadja, vagy nem, ez érthető, hiszen egy szoftvert akár több millióan is megvásárolhatnak. A felhasználás feltételei öt fő téma köré csoportosulnak:

1. Pénzért, vagy ingyen hozzá lehet-e jutnia szoftverhez.
2. A szoftver másolása, terjesztése.
3. A használat időbeli korlátai.
4. A szoftver forráskódja megismerhető, módosítható-e.
5. A szoftver átdolgozásával létrejött újabb szoftver felhasználásának feltételei.

A szoftverlicence típusok:

- Freeware: Ingyen, szabadon felhasználható és terjeszthető a szoftver, viszont annak forráskódját a szerző nem teszi közzé, és azon, ha valahogyan mégis hozzájut valaki, tilos módosítani. Később a szoftver gyártója más kategóriába sorolhatja a termékét.
- Semi-free software: Megkülönböztetik a szoftver üzleti és magán vagy oktatási célú felhasználását. Utóbbiak ingyen használhatják, míg előbbieknek meg kell vásárolniuk a terméket. A forráskód nem nyílt, és nem fejthető vissza.
- Free software: A nyílt forrású szoftverek egyik fajtája. A szerző közkincsé teszi a program forráskódját, engedélyezi az abban való módosítást, a szoftver ingyenes, szabad felhasználását és továbbadását. A legismertebb ilyen típusú licence a GPL.
- Trial: Három hónapos próbaverzió. Leggyakrabban 90-en napig ingyenesen használható a program, de utána csak regisztrálás, vagy a teljese termék megvásárlása esetén. A trial verziók szabadon terjeszthetőek, de forráskódjuk nem nyílt.
- Shareware: A szoftver ingyen használható, szabadon terjeszthető, de csak korlátozott ideig és/vagy mértékben. Lényege, hogy a felhasználó kipróbálhatja, megismerheti a programot, de ha annak teljes verzióját, minden funkcióval együtt használni akarja, akkor meg kell vásárolnia a terméket. Gyakori az időbeli korlátozás is, jellemzően 30 napig áll a felhasználó rendelkezésére ingyen a program.
- Abandonware: Nem hivatalos jogi státusz. Arról van szó, hogy egy szoftver elavult és ennek következtében a gyártó vélhetően nem él a másolás, vagy felhasználás korlátozásának a jogával.

- Public Domain: A szoftver szerzője lemondott mindenféle jogáról a szoftverrel kapcsolatban és azt a közösségre ruházta át. A szerző utólag ezt a döntését nem változtathatja meg.

A licence szerződés rögzíti a terméktámogatás módját (frissítések, javítások, esetleg új verzió), időtartamát. A gyártó által vállalt jótállás és szavatosság információ is a licence szerződésben található.

Bizonylatolás (certification): Jóllehet a tervezés és fejlesztés teljes időtartama alatt figyelembe kell venni a hatósági előírásokat, mégis a tesztelés utolsó mozzanataként, meg kell vizsgálni a létrehozott informatikai rendszert, hogy megfelel-e a hatóságok által állított szabványnak. Ennek igazolására bizonylatot állítanak ki.

Felhasználói dokumentáció:

Felhasználói kézikönyv (instruction manual): Mindazon leírások és magyarázatok összessége, amelyek leírják mire és hogyan használható a rendszer. Először is a specifikáció lényege szerepel benne, vagyis leírjuk milyen szolgáltatásokat tartalmaz az informatikai rendszer. Ne feledjük szoftver és hardver része is lehet! A következő rész az üzembe helyezés, szoftver esetén installálás lépésről-lépésre, lehetőleg képekkel ellátott leírása. Úgy kell megfogalmazni, hogy egy laikus is megértse. Ezután a rendszer összes funkcióját be kell mutatni képernyőképekkel együtt. A felhasználói kézikönyv elengedhetetlen része a rendszer hardver és szoftver követelményeinek a megadása. Szokás egy minimumot meghatározni, amivel már ha lassan, de fut a rendszer, és egy optimálisat. A felhasználói kézikönyv végén általában a vevőszolgálat elérhetőségeit is megtaláljuk.

Biztonsági követelmények dokumentációja: Egy olyan előírás, amely meghatározza mit szabad és mit nem szabad megengedni a rendszer működése során a rendszer szabályos működése érdekében.

SZAKMAI INFORMÁCIÓTARTALOM

ÜZEMELTETÉS, TOVÁBBFEJLESZTÉS, KARBANTARTÁS (EVOLÚCIÓ), AZ ÁTADÁS DOKUMENTUMAI

A rendszer megrendelőnek történő átadásával a rendszer életében új szakasz kezdődik. Nem csupán a rendszert adjuk át a felhasználónak, hanem a **felhasználói kézikönyvet,** az **üzembe helyezési kézikönyv**et, amelyet magában foglalhat a felhasználói kézikönyv is, és a licence szerződést. Az átadásról, ha komplexebb rendszerről van szó (pl. egy egyetem tanulmányi és információs rendszerének átadása esetén készítünk, egy játékprogram boltban vásárlása esetén nem készítünk), **átadás-átvételi jegyzőkönyvet** vezetünk.

Amennyiben a megrendelő igényeli, puszta átadás helyett a **fejlesztők üzembe is helyezhetik a rendszert**. Ez esetben először ellenőrizzük, hogy a körülmények megfelelőek-e, telepítjük a hardvereket, majd a szoftvereket. Ha úgy tűnik minden rendben, akkor kísérleti futtatást végzünk. Az átállás háromféleképpen történhet, hogyha volt korábbi rendszer:

1. **Egy időpontban**, ez esetben, amikor az új rendszer életbe lép, a régit üzemen kívül helyezzük, és minden abban tárolt adatot átviszünk az új rendszerbe.
2. **Párhuzamos működéssel**, ez esetben a régi rendszert változatlanul futtatjuk, és üzembe helyezzük mellette az újat, a kettő egy ideig egyszerre fut.
3. **Szakaszos átállással**, hasonló az előző esethez, mert itt is két rendszer fut egyszerre, viszont az új rendszer egyre több feladatot vesz át, azaz egy időben a feladatok egy részét a régi rendszer, másik részét az új rendszer végzi el, mígnem az összes feladatot átveszi az új rendszer.

Az üzembe helyezéssel párhuzamosan, vagy azt követően a fejlesztő cég gyakran **oktatás**t is tart az új rendszer funkcióiról, használatáról, a megrendelő igényeinek megfelelően. Átadáskor szükség lehet az **adatok átvitelére** a régi rendszerből az újba, szükség esetén ezt is végzhetik.a fejlesztők.

A megrendelőnek vagy felhasználónak történő átadás után nem ért véget a munka. A rendszer **karbantartás, frissítés** feltételeit és időtartamát a licence szerződés tartalmazza. Mint, mondtuk a tesztelés nem tud minden hibát kiszűrni, előfordulhat, hogy olyan új bemeneteket kap a rendszer, amelyekkel hibás működés lép fel. Ez esetben gyors **javításra, korrekcióra** van szükség. Ennek megelőzésére sok rendszerfejlesztő, szoftvergyártó cég a meglévő rendszerekhez folyamatosan biztonsági és egyéb **frissítéseket** készít és tesz közé a felhasználói számára. A **javító csomagokat** találóan **patch**nek nevezzük, ennek a szónak az eredeti jelentése sebtapasz, folt. A patchek általában kis méretűek. A frissítéseket legtöbbször a rendszer teljes változatának birtokában lehet használni és ezeket ingyen letöltheti a felhasználó. A frissítésnek van egy másik jelentése is, ugyanis vannak szoftverek, amelyek csakis akkor érnek valamit, ha naprakészek, például a különböző térkép adatbázisok.

Számos okból szükség lehet arra, hogy egy meglévő rendszerhez egy új szoftver vagy hardver elemet illesszenek, ennek elvégzésére, amennyiben nehézségekbe ütközik a rendszer sajátosságai miatt, az adaptációra felkérhetik a rendszer fejlesztőjét.

Végül a felhasználó igényei, vagy nagyobb technológiai fejlesztés következtében a rendszer nagyobb volumenű karbantartására, ún. **rendszerkorrekcióra** is szükség lehet.

Összefoglalás

Határozza meg a rendszer fogalmát!

Mit nevezünk informatikai rendszernek?

Sorolja fel az informatikai rendszer létrehozásának főbb lépéseit!

Milyen életciklus modelleket ismer?

Miben különbözik a fejlesztési fázist a definíciós fázistól?

TANULÁSIRÁNYÍTÓ

Az imént ismertetett tervezési lépések a következő három fejezet alapját is képezik, tehát ezek megértése és megtanulása elkerülhetetlen. A lépések egymásra épülnek. A tanulás során javaslok egy olyan sorrendet betartani, ami a tananyag ismertetésekor is történt. Ha tisztában vagyunk az alapfogalmakkal és a tervezés alaplépéseivel, akkor érdemes gyakorolni a megvalósításukat pár feladaton keresztül, ami a következő részben megtalálható lesz!

. ÖNELLENÖRZŐ FELADATOK

1. Meg kell terveznie egy macskatápszer gyártó cég honlapját. A honlap kicsi, a követelmények rögzítettek, az ön csapata három fős lesz. Melyik életciklust modell mentén készítené el a weblapot? Magyarázza meg választását!

2. Felkérik, hogy tervezzen meg egy interaktív alapfokú angol nyelvvizsgára felkészítő programot. Hogyan tesztelné az elkészült szoftvert? Miért?

3. Ön egy shareware programot használ, amely 30 napos időkorláttal vehet igénybe. Milyen módon használhatja a 30 nap letelte előtt és után a programot?

MEGOLDÁSOK

1. feladat

Alapvetően a prototípus, vagy a vízesés modell jöhet szóba. A prototípus modell gyakorlat orientált, míg a vízesés modellben nagy hangsúlyt kap a dokumentáció is. Mindkettő alkalmas rövid időtartalmú, egyszerű, kevés létszámú informatikai projektek irányítására, viszont a vízesés modell nem alkalmas arra, hogy a projektek kezdése után nem sokkal legen egy működő rendszer, amit a megrendelőnek be lehet mutatni. Ez persze nem is feltétlenül szükséges, viszont a folyamatos egyeztetés a megrendelővel, nagyobb eséllyel eredményez egy olyan projekt eredményt, amellyel a megrendelő maximálisan elégedett lesz. Valamint úgy vélem, ez esetben a dokumentáció nem hangsúlyos, sokkal inkább az elkészült mű. Természetesen a feladatok kiosztása többféle metódus szerint történhet, attól függően milyen szakterülettel rendelkeznek a kollegák, mivel ezt nem ismerem, ezért ennek elemzésébe nem megyek bele.

2. feladat

Többféle tesztet is futtatnék rajta, dinamikusat és statikusat is. A dinamikusok közül a fekete doboz módszerrel kezdeném, hiszen egy ilyen szoftver nem rendelkezik túl sok funkcióval (hanem egy viszonylag kevés számút ismétel), viszont nagyon fontos, a helyes, hibamentes működés mellett, hogy könnyen használható, ergonomikus kezelői felületű legyen. Tehát fekete doboz módszerrel tesztelném a programot. A kevés funkcióval egyszerűbb szerkezet is párosul, ezért, ha megtaláljuk a hibát, akkor, jóllehet a teszt erre nem alkalmas, de mégis nagy valószínűséggel a hiba helyét is felelhetjük. A szoftver statikus tesztelését is szükségesnek tartom, méghozzá két féle szempontból. Egyrészt a forráskód átnézését egy programozó, másrészt a szakmai, angol nyelvi tartalmat egy angol nyelvtanár által. Végül stressz teszt elvégzése is elengedhetetlen, hiszen a diákok nem feltétlenül a leírás szerint fogják használni a programot.

3. feladat

A 30 nap letelte előtt a program minden funkcióját szabadon használhatom. Az egyetlen kellemetlenség a használat során, hogy a program minden induláskor figyelmeztet, hogy még hány nap van hátra a 30 napos időkorlátból. Viszont nem fejthetem vissza a forráskódot, nem módosíthatok a szoftveren. A 30 napos próbaverziót szabadon terjeszthetem is. A 30 nap letelte után azonban a szoftver nem használható tovább, addig, ameddig valamiképpen nem vásárolom meg a terméket. A vásárlás bizonyításaként általában valamilyen szériaszámot kér a program, és ennek helyessége esetén már szabadon használható a szoftver ugyanúgy, ahogyan a 30 nap letelte előtt.

IRODALOMJEGYZÉK

FELHASZNÁLT IRODALOM

AVINCULUI Mihály: *Informatikai rendszerek tervezése és menedzsmentje.* Kolozsvár (Ábel) 2010.

CHORMEN, Thomas- LEISERSON, Charles- RIVEST, Ronald- STEIN, Clifford: *Új algoritmusok*. Budapest (Scolar) é.n.

JÁRDÁN Tamás- POMAHÁZI Sándor: *Adatszerkezetek és algoritmusok*. Eger (Líceum) 2010.

LOVÁSZ Péter: *Operációs rendszerek*. Békéscsaba (Booklands) 2006.

AJÁNLOTT IRODALOM

KNUTH, Donald: *Alapvető algoritmusok*. Budapest (Műszaki) 1994.

LIPSCHUTZ, Seymour: *Adatszerkezetek*. Budapest (Panem-McGraw-Hill) 1993.

ROLLAND, Fred: *Adatbázisrendszerek.* (Panem) 2002.

SIMMERVILLE, Ian: *Szoftverrendszerek fejlesztése*. (Panem) 2007.

TANENBAUM, Andrew: *Számítógép architektúrák*. (Panem) 2001.