Dr. Iszály György Barna

# Programozás módszertana

# Mi is az a szoftver?

- Számítógépes program, vagy programok halmaza, mely kiegészül kiegészül a használatát lehetővé tevő dokumentációkkal, konfigurációs adatokkal, információs webhelyekkel.
- Egy termék - valamilyen hasznos célt kell kiszolgálnia, elő kell állítani, az előállítási folyamatnak költségei és időkorlátai vannak, ezért a felhasználó hajlandó fizetni érte, cserébe valamilyen minőséget vár el
- A szoftvernek azonban a más ipari termékektől eltérő tulajdonságai is vannak:
  - Nem anyagi jellegű. Létezik ugyan tárgyiasult formája, de az nem más, mint adathordozók és dokumentációk halmaza.
  - Nincsenek egyedi példányai. Egy szoftvert lemásolhatunk tetszőleges példányban, de ezek a másolatok teljes mértékben egyenértékűek az eredetivel. Ennek következménye, hogy előállítása nem igényli a tervezés – sorozatgyártás fázisokat.
  - Tükröznie kell a valóságot, de nem fizikai törvények vonatkoznak rá.
  - Bonyolultsága (komplexitása) a legtöbb ipari terméket messze meghaladja.
  - Ezeket a speciális tulajdonságokat kell figyelembe venni, ha a szoftver előállításának technológiáját vizsgáljuk.

# Szoftver technológia

- Az IEEE szervezet 1983-as definíciója szerint:
- „The technological and managerial discipline concerned with systematic production and maintenance of software products that are develop and modified on time and within cost estimated.”
- „Technológiai és vezetési alapelvek, amelyek lehetővé teszik programok termékszerű gyártását és karbantartását a költség- és határidő korlátok betartásával.”
- Egyértelműen mérnöki tevékenységnek minősíti a szoftver előállítását: a gyártási folyamatot (általában szigorú) idő- és költség korlátok betartása mellett kell lebonyolítani!

# A szoftverfejlesztés makrofolyamata

- Probléma, kérés, ötlet ( a megrendelőtől )
- A felvetés értelmezése, pontos megfogalmazása, feladattá alakítása
- Döntés arról, hogy érdemes-e programot írni
- A program megírásához használható erőforrásaink számbavétele (hardver, szoftver)
- A feladat megoldásához szükséges INPUT (bemenő) adatok számbavétele
- Az OUTPUT (kimeneti) információk tartalmának és formájának megtervezése (képernyő és nyomtatási kép)
- A megoldás lépésről lépésre történő megadása ( az algoritmus megtervezése)
- Az algoritmus leírása (folyamatábra, struktogram, leírás, pszeudokód)
- A programnyelv kiválasztása
- Az algoritmus lekódolása ( a program megírása )
- A program tesztelése (próbaadatokkal történő kipróbálása)
- A dokumentáció elkészítése
- Esetleg árualakra hozás

# A szoftverfejlesztés életciklusának általános feladatai

- Követelmények megfogalmazása - funkcionális specifikáció
- Rendszertervezés (design) - rendszerterv
- Kódolás, testreszabás, tesztelés és dokumentálás
- Bevezetés

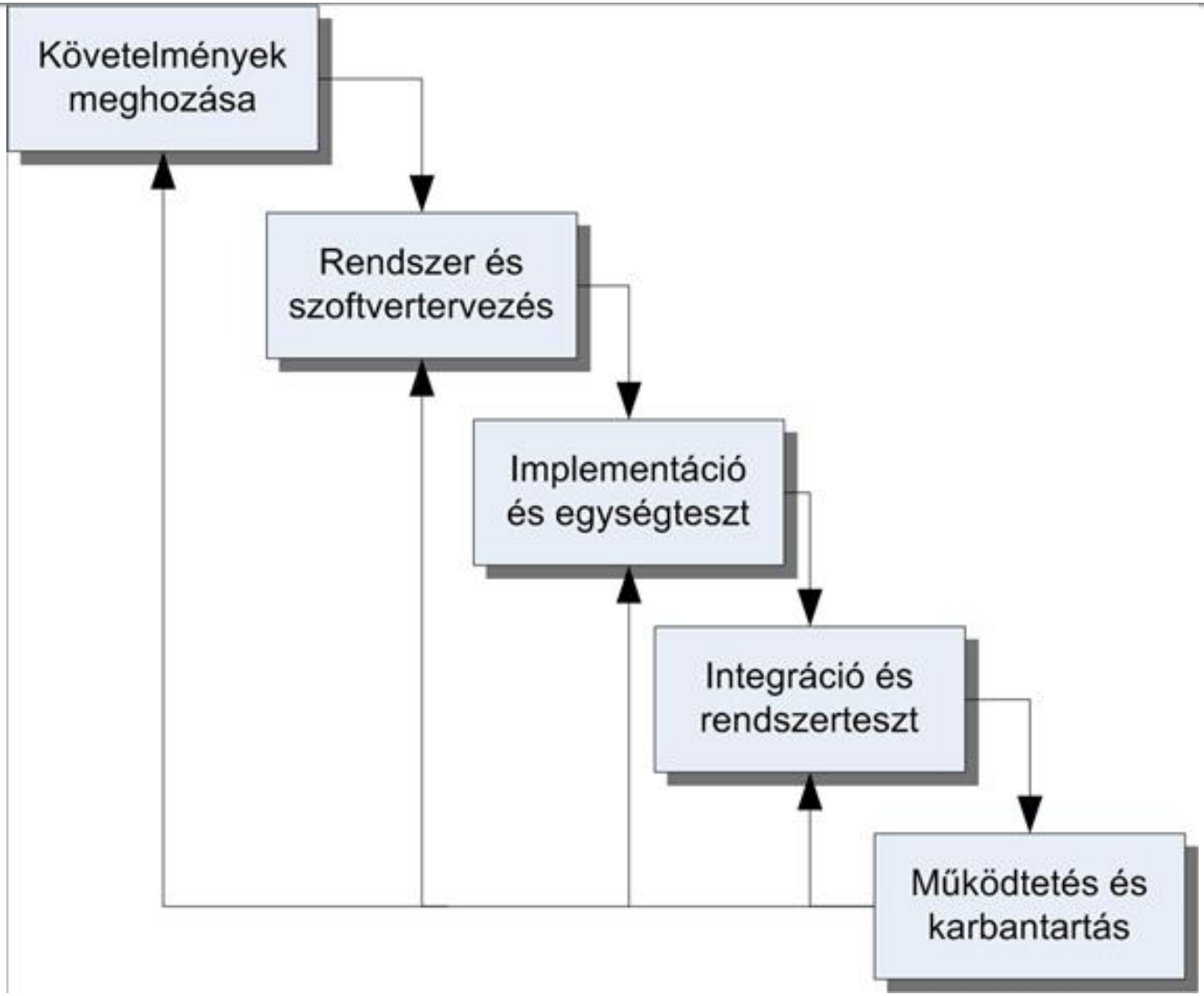
# Miért kell tervezni?

- Megbízhatóság
  - Ha egy programot megfelelően terveztünk meg, akkor elvileg megfelelően kell működnie
- Költségek
  - A jó terv lecsökkenti az idő-, és ezáltal költségigényes, tesztelési fázist
- Karbantartás
  - A jól megtervezett programot egyszerűbb módosítani

# Szoftverfejlesztési modellek

1. Vízesésmodell
2. Evolúciós vagy iteratív fejlesztés
3. Komponens alapú fejlesztés

# Vízesésmodell



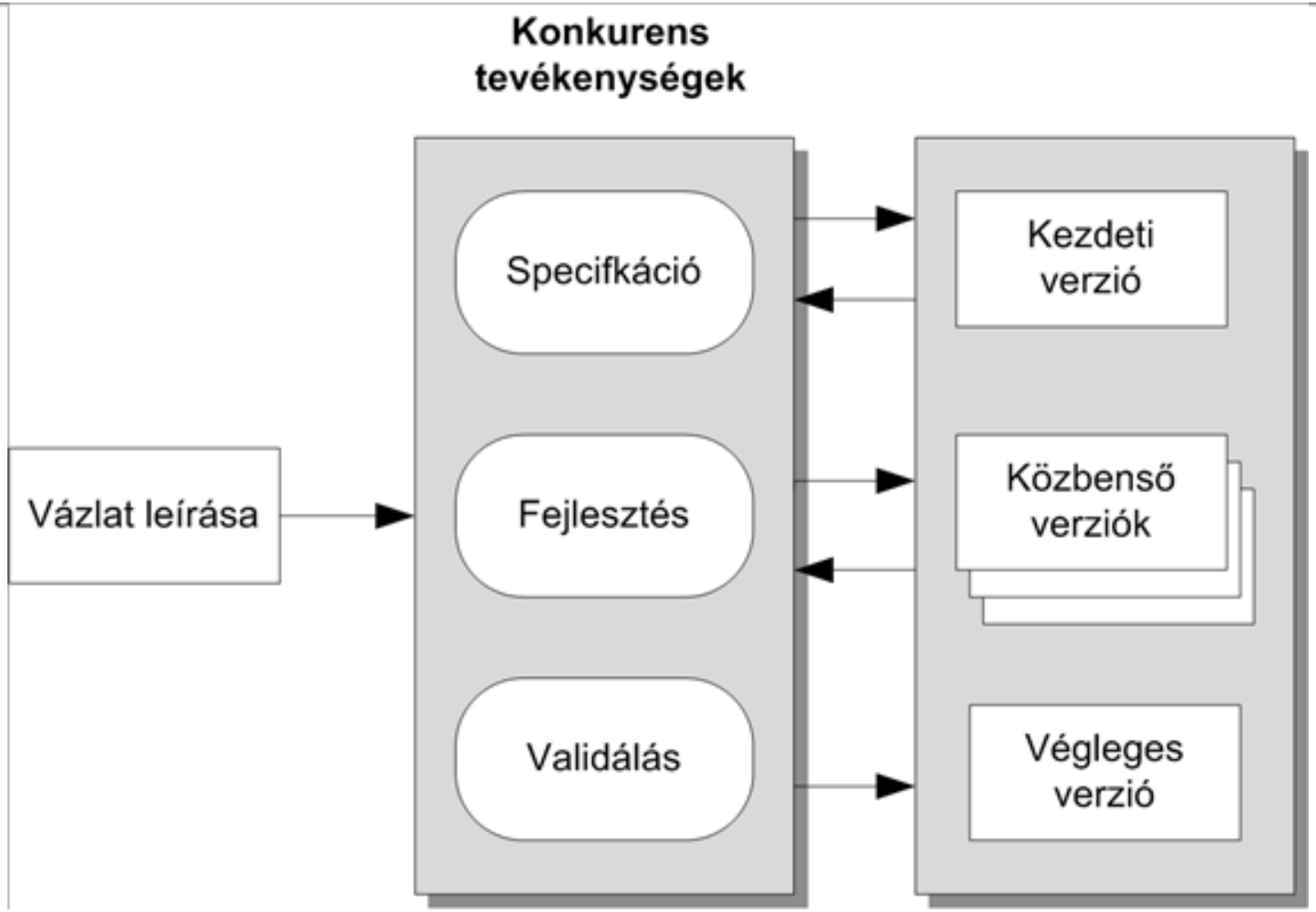
# Vízesésmodell

- **Követelmények elemzése és meghozása**: a rendszer szolgáltatásai, megszorításai és célja a rendszer felhasználóival történő konzultáció alapján alakul ki. Ezeket később részletesen kifejtik, és ezek szolgáltatják a rendszer specifikációt.
- **Rendszer- és szoftvertervezés**: szét választódnak a hardver- és szoftverkövetelmények. Itt alakítjuk ki a rendszer átfogó architektúráját. A szoftver tervezése az alapvető szoftverrendszer-absztrakciók, illetve a közöttük levő kapcsolatok azonosítását és leírását is magában foglalja.
- **Implementáció és egységteszt:** a szoftverterv programok, illetve programegységek halmazaként realizálódik. Az egységteszt azt ellenőrzi, hogy minden egység megfelel-e a specifikációjának.
- **Integráció és rendszerteszt**: a különálló programegységek, illetve programok integrálása és teljes rendszerként való tesztelése, hogy a rendszer megfelel-e a követelményeknek. Ez után a szoftverrendszer átadható az ügyfélnek.
- **Működtetés és karbantartás**: általában ez a leghosszabb fázis. Megtörtént a rendszertelepítés és megtörtént a rendszer gyakorlati használatbavétele. A karbantartásba beletartozik az olyan hibák javítása, amelyekre nem derült fény az életciklus korábbi szakaszaiban, a rendszeregységek implementációjának továbbfejlesztése, valamint a rendszer szolgáltatásainak továbbfejlesztése a felmerülő új követelményeknek megfelelően.

# Vízesésmodell

- A fázisok eredménye egy dokumentum.
- A következő fázis addig nem indulhat el, amíg az előző fázis be nem fejeződött.
  - A gyakorlatban persze ezek a szakaszok kissé átfedhetik egymást.
- Maga a szoftverfolyamat nem egyszerű lineáris modell, hanem a fejlesztési tevékenységek iterációjának sorozata. Ez a vízesésmodellnél a visszacsatolásokban jelenik meg.
- Probléma: a projekt szakaszainak különálló részekké történő nem flexibilis partícionálása. Egy komplex, bonyolult probléma megoldása nem végezhető el hatékonyan ezzel a megközelítéssel. Csak akkor használható jól, ha már előre jól ismerjük a követelményeket, melyeket részletes és pontos specifikáció követ.

# Evolúciós fejlesztés



# Evolúciós fejlesztés

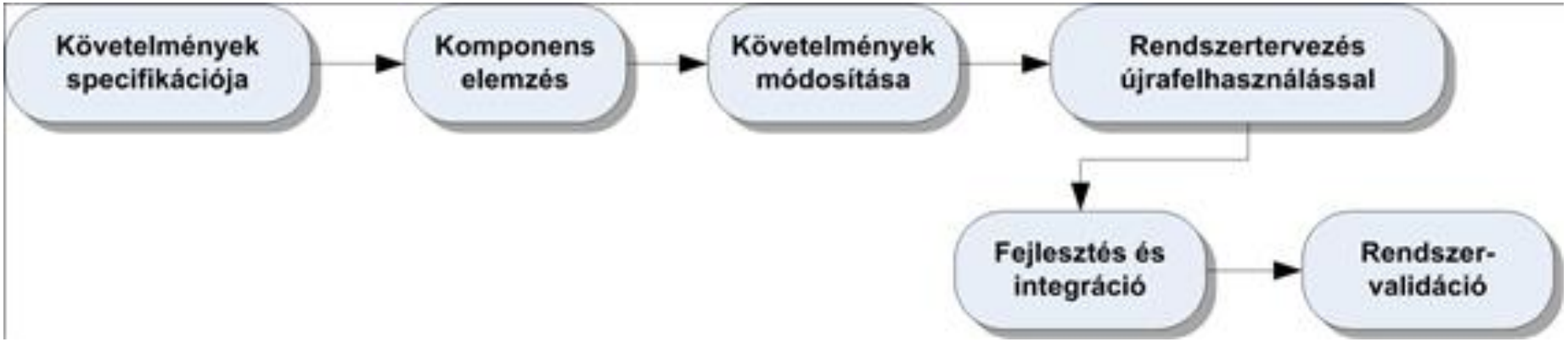
- A fejlesztőcsapat kifejleszt egy kezdeti implementációt, majd azt a felhasználókkal véleményezteti
- Addig finomítják, amíg a megfelelő rendszert el nem érik.
- Ez sokkal inkább érvényesíti a tevékenységek közötti párhuzamosságot és a gyors visszacsatolásokat.
- Két típusát különböztetik meg:
  - Feltáró fejlesztés: célja egy működőképes rendszer átadása a végfelhasználóknak. Ezért elsősorban a legjobban megértett és előtérbe helyezett követelményekkel kezdik a fejlesztés menetét. Ennek érdekében a megrendelővel együtt tárjuk fel a követelményeket, és alakítják ki a végleges rendszert, amely úgy alakul ki, hogy egyre több, az ügyfél által kért tulajdonságot társítunk a már meglévőkhöz. A kevésbé fontos és körvonalazatlanabb követelmények akkor kerülnek megvalósításra, amikor a felhasználók kérik.
  - Eldobható prototípus készítés: a fejlesztés célja ekkor az, hogy a lehető legjobban megértsük az ügyfél követelményeit, amelyekre alapozva pontosan definiáljuk azokat. A prototípusnak pedig azon részekre kell koncentrálni, amelyek kevésbé érthetők.

# Evolúciós fejlesztés

- Két probléma:
  - **A folyamat nem látható**:
    a menedzsereknek rendszeresen szükségük van leszállítható részeredményekre, hogy mérhessék a fejlődést.
  - **A rendszerek gyakran szegényesen strukturáltak**:
    a folyamatos változtatások lerontják a rendszer struktúráját, így kevésbé érthetővé válik. A szoftver változásainak összevonása pedig egyre nehezebbé és költségesebbé válhat.

- A várhatóan rövid élettartalmú kis vagy közepes rendszerek esetén célszerű az alkalmazása
- Max. 500.000 programsorig
- Hosszú élettartalmú rendszerek esetén az evolúciós fejlesztés válságossá válhat pontosan az evolúciós jellege miatt.

# Komponens alapú fejlesztés



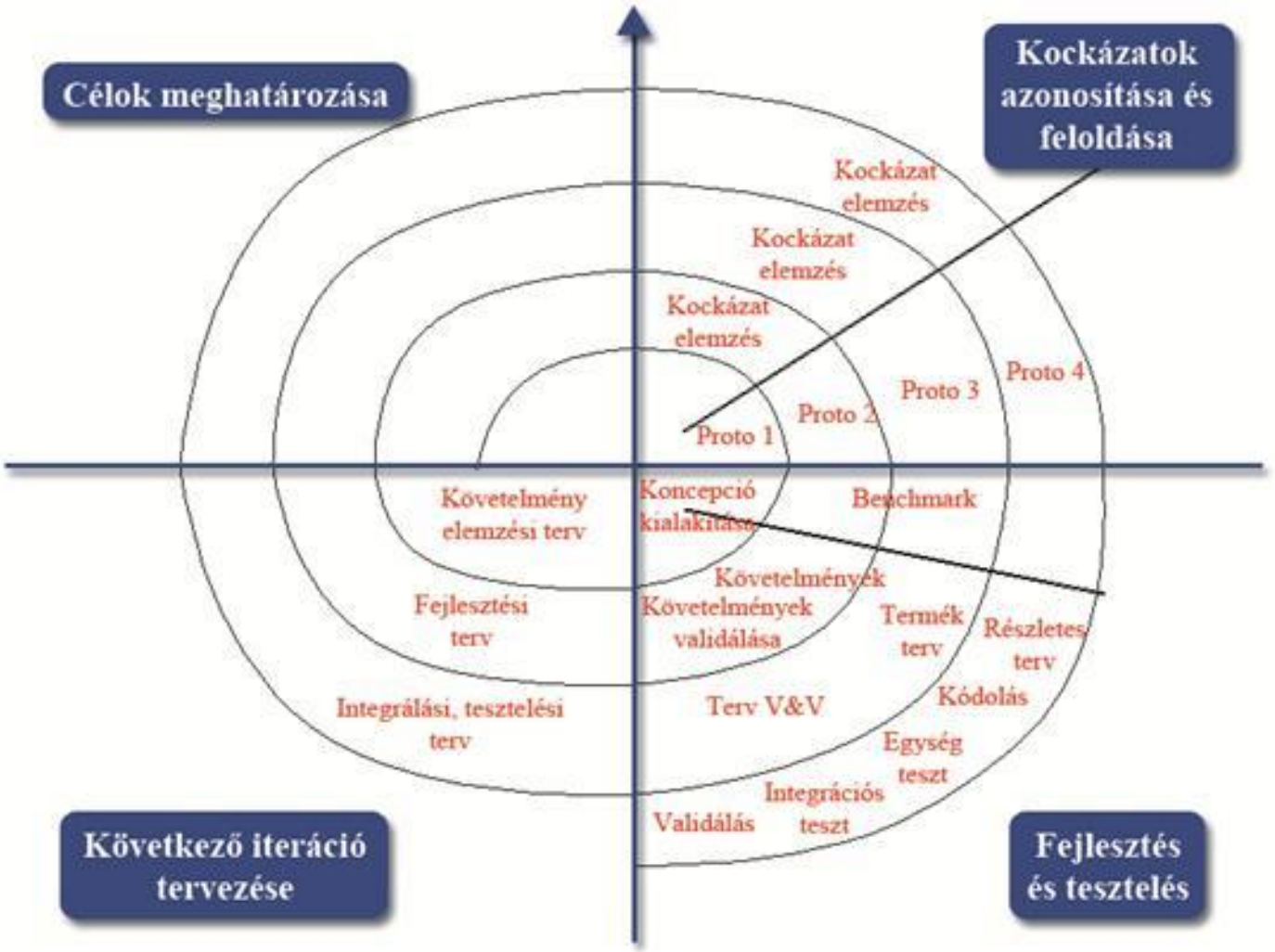
# Komponens alapú fejlesztés

- Az újrafelhasználás-orientált megközelítési mód nagymértékben az elérhető újrafelhasználható szoftverkomponensekre, illetve azok egységes szerkezetbe történő integrációjára támaszkodik.
- Az eddigiektől eltérő fázisok a következőek:
  - **Komponenselemzés**: az adott követelményspecifikációnak megfelelően megkeressük azon komponenseket, amelyek megvalósítják azok funkcióit, implementálták azokat. A legtöbb esetben nincs egzakt illeszkedés, a felhasznált komponens a funkciók csak egy részét nyújtja.
  - **Követelménymódosítás**: a fázisban a megtalált komponensek információit felhasználva elemezzük a követelményeket, majd módosítani kell azokat az elérhető komponenseknek megfelelően. Ahol nem lehetséges a követelmény módosítása, ott újra el kell végezni a komponenselemzést és alternatív megoldást kell keresni.
  - **Rendszertervezés újrafelhasználással**: ebben a szakaszban a rendszer szerkezetének tervezését hajtjuk végre. A tervezés kulcseleme az, hogy milyen komponenseket akarunk újrafelhasználni, és úgy alakítani a szerkezetet, hogy azok működhessenek. Amennyiben nincs elérhető újrafelhasználható komponens, akkor új szoftverek is kifejleszthetők.
  - **Fejlesztés és integráció**: a nem megvásárolt, illetve átalakításra kerülő komponenseket ki kell fejleszteni és a rendszerbe integrálni. A rendszer-integráció ebben a modellben sokkal inkább tekinthető a fejlesztési folyamat részének, mint különálló tevékenységnek.

# Komponens alapú fejlesztés

- Előnye:
  - csökkenti a kifejlesztendő szoftverek számát a komponensek újrafelhasználásával
  - közvetve a költségeket redukálja
  - Redukálja a felmerülő kockázati tényezőket
  - Így gyorsabban is leszállítható a megrendelőnek a program.
- Hátránya:
  - a követelményeknél elkerülhetetlenek a kompromisszumok
  - az elkészült rendszer nem felel meg a megrendelő valódi kívánságainak.

# Spirális fejlesztési modell



# Spirális fejlesztési modell

- A szoftverfolyamatot nem tevékenységek és közöttük található esetleg visszalépések sorozataként tekinti, hanem inkább egy spirálként reprezentálja
- A spirál minden egyes körben a szoftverfolyamat egy-egy fázisát reprezentálja
- Tetszőleges számú kör, mint iteráció tehető meg
- A körök:
  - Legbelső kör - a megvalósíthatósággal foglalkozik
  - Következő kör - a rendszer követelményeinek meghatározásával foglalkozik
  - Ezt következő kör - a rendszer tervezésével foglalkozik
  - Stb.

# Spirális fejlesztési modell

- A spirál minden egyes ciklusát négy fő szektorra oszlik:
  - **Célok kijelölése**: az adott projektfázis által kitűzött célok meghatározása. Azonosítani kell a folyamat megszorításait, a terméket, fel kell vázolni a kapcsolódó menedzselési tervet. Fel kell ismerni a projekt kockázati tényezőit, és azoktól függően alternatív stratégiákat kell tervezni ha lehetséges.
  - **Kockázat becslése**: minden egyes felismert kockázati tényező esetén részletes elemzésre kerül sor. Lépéseket kell tenni a kockázat csökkentése érdekében.
  - **Fejlesztés és validálás**: a kockázat kiértékelése után egy fejlesztési modellt kell választani a problémának megfelelően. Pl. evolúciós, vízesés, stb modellek.
  - **Tervezés**: A folyamat azon fázisa, amikor dönteni kell arról, hogy folytatódjon-e egy következő ciklussal, vagy sem. Ha a folytatás mellett döntünk, akkor fel kell vázolni a projekt következő fázisát.

# A programkészítés folyamata

- ***Specifikáció***: a feladat pontos meghatározása
- ***Tervezés***: hogyan kell megoldani a feladatot?
- ***Kódolás***: adott programozási nyelven megírt program
- ***Tesztelés***: jó vagy rossz a program?
- ***Hibakeresés, -javítás***: hol a hiba? → teszteléshez vissza → … → **helyes program**
- ***Hatékonyság vizsgálat***: minőségvizsgálat és javítás → **jó program**
- ***Dokumentálás***: felhasználói, fejlesztői → **használható program**
- (***Karbantartás***: a változó igényeknek megfelelő fejlesztések → **időtálló program**)

# Specifikáció

- a programkészítés folyamatának első lépése
- a feladat pontos meghatározása
- olyan leképezést (programfüggvényt) határoz meg, amely a bemenő értékekhez hozzárendeli a jó eredményt.
- a feladat szöveges és formalizált, matematikai leírásán túl tartalmazza a megoldással szemben támasztott követelményeket, környezeti igényeket is.

**Absztrakt specifikáció**:

- **Elő-, utófeltételek**: A bemenő-, illetve a kimenő adatokra kirótt feltételek.

# Specifikáció részei

**Feladatspecifikáció:**

1. feladat szövege
2. a bemenő és kimenő adatok elnevezése, értékhalmazának leírása (vagyis az ***állapottér*** meghatározása)
3. a feladat szövegében használt fogalmak definíciói, eredmény kiszámítási szabálya
4. a bemenő adatokra meghatározott előfeltételek
5. a kimenő adatokra meghatározott utófeltételek

**A program specifikálása:**

1. program elő- és utófeltétele
2. a bemenő és a kimenő adatok elnevezése, értékhalmazának leírása

# Technikai specifikáció

1. a feladat szövegében használt fogalmak definíciói
2. a program környezetének leírása: *hardver-, memória- és perifériaigény, programozási nyelv, szükséges fájlok stb.*
3. a programmal szembeni egyéb követelmények: *minőség, hatékonyság, hordozhatóság stb.*

**Követelmények a specifikációval szemben**: (szöveges vagy formális specifikációval szemben egyaránt)

- egyértelmű, pontos, teljes
- rövid, tömör, formalizált
- szemléletes, érthető

# Feladatspecifikációs példa

Egy mezőn ***N*** ló és ***M*** szekér található. Az ***i.*** ló ereje $p_i$, az ***i.*** szekér súlya $g_i$. Egy ló maximum egy szekeret húzhat, de ennek a súlya kisebb vagy egyenlő a ló erejével.

**Követelmény**

Határozzuk meg a *lehetséges legnagyobb ló-szekér párt* úgy, hogy a párban szereplő ló legyen képes húzni a megfelelő szekeret.

**Bemeneti adatok**

A **lovak.in** bemeneti állomány első sorában $N + 1$ természetes szám található: $N\ p_1\ p_2\ \ldots\ p_N$, egy-egy szóközzel elválasztva, a fenti szövegben leírt jelentésekkel. A második sorban, hasonlóan, $M + 1$ természetes szám található: $M\ g_1\ g_2\ \ldots\ g_M$, egy-egy szóközzel elválasztva.

# Feladatspecifikációs példa

**Kimeneti adatok**

A **lovak.out** kimeneti állományba *egyetlen számot* írunk, a kért legnagyobb lehetséges ló-szekér párok számát.

**Megszorítások és pontosítások**

- $1 \le N, M \le 10\,000$
- $1 \le p_i < 2^{15}$, $i = 1,2,\ldots,N$
- $1 \le g_i < 2^{15}$, $i = 1,2,\ldots,M$
- A tesztek 50%-ban $M, N \le 1000$

**Példa**

| lovak.in | lovak.out | Magyarázat |
|---|---|---|
| 6 3 3 4 4 5 1<br>6 3 4 5 2 1 6 | 5 | A párok (ló indexe, szekér indexe):<br>(1, 1), (2, 4), (3, 2), (5, 3), (6, 5). |

**Maximális végrehajtási idő/teszt**: **0.1** mp/teszt

**Rendelkezésre álló memória: 16** Mb

# Mit értünk programtervezésen?

- A leendő program ***szerkezetének*** megalkotása
  - A program elemeinek megfelelő sorrendben való ***összerakása***
  - A szerkezetnek több szintje lehet
    - ***átfogó szerkezet:*** függvények, eljárások, alprogramok
    - ***finomszerkezet:*** változók, műveletek (utasítássorok, ciklusok, döntési szerkezetek)
  - Elkülönül a kódírástól (kódolástól)

# A programtervezés lépései

- A probléma meghatározása, felmérése: ***specifikáció***

***Részei:***

- Bemeneti adatok: értékhalmaz, méret + tulajdonságok, korlátozó tényezők – (***előfeltételek***)
- Eredmények (tulajdonságok, a megoldással szembeni követelmények) – (***utófeltételek***)

***Tulajdonságai:***

- Egyértelmű, pontos, teljes
- Rövid, tömör, formalizált
- Szemléletes, érthető

# A terv megvalósítása

**Algoritmustervezés**

- Eszközei: algoritmusleíró eszközök
  - ***Folyamatábra***
  - Struktogram
  - Leírás mondatszerű elemekkel (***pszeudokód***)

**Paradigmaválasztás**

- strukturált?
- objektumorientált?
- funkcionális?
- stb.

# Programkészítési elvek

***Stratégiai elvek***:

- a feladatot néhány részfeladatra bontjuk → lépésenkénti finomítás →
- felülről lefelé való kifejtés: először átfogóan oldjuk meg a feladatot, majd a részfeladatokat addig finomítjuk, amíg olyan utasítások szintjéig nem érünk, amelyeket a gép már végre tud hajtani (piramis-elv)
- alulról felfelé való építés: a piramist fordítva építjük fel

# Taktikai elvek

- ***párhuzamos finomítás elve*:** egy adott szint minden részfeladatát finomítjuk, nem szaladunk előre egy könnyűnek vélt ágon
- ***döntések elhalasztásának elve*:** célszerű minél későbbre halasztani azokat a döntéseket, amelyek a gép ill. a programnyelv adottságainak kihasználási módját jelentenék
- ***vissza az ősökhöz elv*:** ha zsákutcába kerülünk, vissza kell lépni az előző szintre

# Taktikai elvek

- ***adatok leszigetelésének elve*:** az egyes programrészeken belül alkalmazott adatokat ne vigyük be más programrészletbe. (Paraméterekkel és lokális változókkal dolgozunk)
- ***párhuzamos ágak függetlenségének elve:*** egy szinten lévő eljárások egymást nem hívhatják, egymás változóit nem használhatják
- ***szintenkénti teljes kifejtés elve:*** a szint leírásának tartalmaznia kell az alatta levő szint eljárásainak specifikációját.

# Technológiai elvek

- ***algoritmus leírási szabályok*:** kevés, de egyértelmű szabályt kell kialakítani. A fő programeszközök (szekvencia, ciklus, elágazás) jól különüljenek el.
- ***világos tagolás*:** csak a szervesen összefüggő utasítások kerüljenek egy sorba
- ***bekezdéses leírás (indentálás)*:** az algoritmus szintekre tagolását a leírás formája is tükrözze
- ***összetett struktúrák zárójelezése*:** ciklus, elágazás, eljárás elejének és végének jelölése (C++: {…}, Pascal, Delphi: begin…end)
- ***beszédes azonosítók elve*:** az azonosító utal arra, hogy mire használjuk

# Technikai elvek

- ***barátságosság, udvariasság*:** tájékoztatás, segítségnyújtás
- ***biztonságosság*:** a felhasználói hibalehetőségekre fel kell készíteni a programot, és lehetőséget adni a javításra
- ***jól olvasható*** program: minél kisebb programegységek
- ***jól dokumentált*** program: kommentek

# Esztétikai, ergonómiai elvek

- ***lapkezelési technika*:** egyszerre egy képernyőnyi információt jelenítünk meg, (egy sor ne legyen szélesebb, mint a képernyő), lapozás biztosítása, nyomtatásban lapszámozás, fejléc, lábléc
- ***menütechnika*:** felhasználóval való párbeszéd biztosítása kényelmes formában
- ***ikontechnika*:** grafikus kis ábrák szemléletesebbek lehetnek a szöveges megjelenítésnél
- ***értelmezési tartomány kijelzése*:** beolvasáskor jelezzük az értéktartományt és a mértékegységet
- ***fontos adatok kiemelése*:** időigényes feladatvégzés során küldjünk üzenetet a felhasználónak, hogy a program éppen hol tart

# Esztétikai, ergonómiai elvek

- ***tördelés*:** a megjelenő szövegben a sorok, szavak tördelése a helyesírás szabályainak megfeleljen
- ***következetesség*:** beolvasáskor, kiíráskor
- ***hibajelzés*:** írjuk ki a hiba okát, adjunk instrukciót a javításra, legyen visszaállítható a hiba előtti képernyő
- ***naplózás*:** a fontos események automatikusan íródjanak fájlba
- ***makrók, funkcióbillentyűk*:** gyors eléréshez bizonyos funkciókhoz rendeljünk billentyűkombinációt
- ***segítség*:** bárhonnan legyen elérhető (- funkcióbillentyűvel)
- ***ablaktechnika*:** az ablakok a képernyő elkülönített részein jelennek meg, ablak: keret+tartalom

# Az adatok megtervezése

- Hol helyezkedhetnek el a program futásához szükséges adatok?
  - Háttértáron: fájlkezelés
  - Operatív tárban (memória, RAM)
- A memóriában elhelyezkedő adatok kezelése
  - Ha egy adat nem változik a program futása során: ***állandó***, ***konstans*** (ha többször használjuk, érdemes nevet adni neki)
  - Ami változik a futás során: ***változó***

# Adatszerkezetek megtervezése

- A feladat követelményeinek és a feldolgozandó adathalmazoknak függvényében:
  - ***Statikus adatszerkezetek:*** tömb, struktúra (rekord), halmaz
  - ***Félstatikus adatszerkezetek:*** verem, várakozási sor, hasító tábla
  - ***Dinamikus adatszerkezetek:*** lista, fa, gráf

# Kódolás

- Az algoritmus ***implementálása***: leírás programozási nyelven
  - a fejlesztői dokumentáció alapján ***szétosztják*** a tervet
  - minden programozó ***teszteli*** a saját munkáját
  - ***összerakják*** a részeket
  - ***rendszertesztelés***
- Felhasználói dokumentáció

# Strukturált programozás

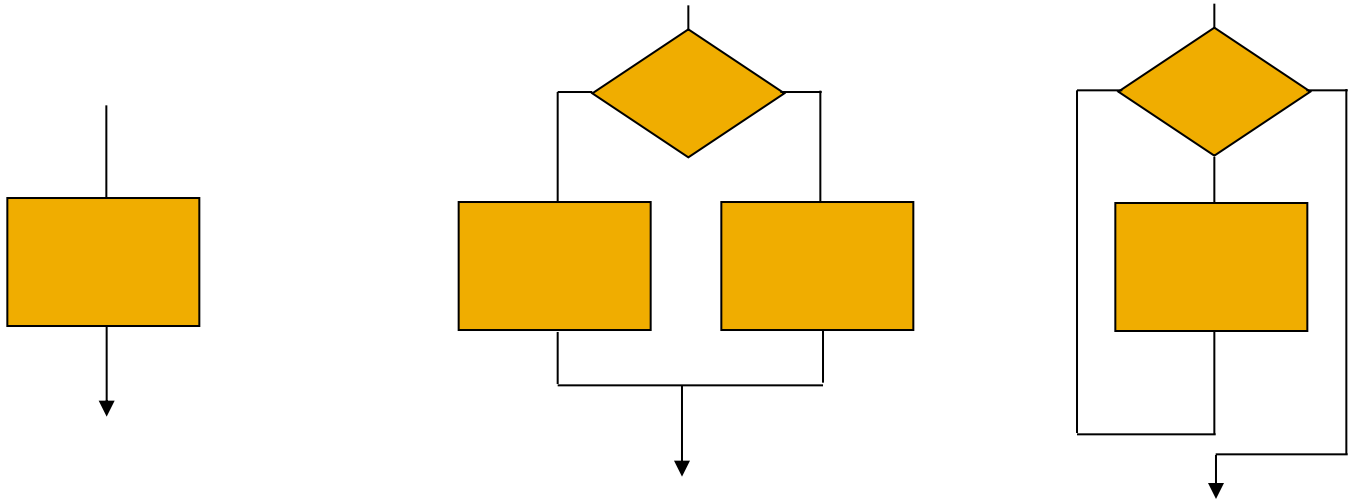
- A strukturált programozás jelenti valamennyi ma használatos programtervezési módszer ***alapját***
- Ma: a strukturált programozás a programfejlesztés ***egyetlen*** lehetséges módja

- ***a nagy problémát kisebb részfeladatok halmazára bontjuk, és a megoldást a részfeladatok megoldásával kezdjük***
  - alkalmazzuk a lépésenkénti finomítás elvét: a megoldandó feladatot részfeladatokra bontjuk úgy, hogy önmagukban megoldhatók legyenek
- ***az összes részfeladat megoldása után, az eredeti feladat megoldása a megfelelő illesztésekre korlátozódik***

# Strukturált programozás

- A **goto** utasítás használata kerülendő
- A strukturált programozás gyakran a ***felülről lefelé*** történő elvonatkoztatást, tervezést is jelenti
- Több szint, egymásba ágyazás.
- Az algoritmus ***alprogramokra*** (részekre) bontása
- ***Böhm, Jacopini***: ***bármely program megírható pusztán az alapstruktúrák használatával***
- A program karbantarthatóságának biztosításához elengedhetetlenül szükséges a strukturált programozás

# Építőelemei



- Minden építőelemnek csak **egy** belépési, illetve kilépési pontja van
- Egy sem tartalmaz háromnál több alapelemet
- A belépési pont az algoritmuselemek elején, a kilépési pont a végén található

# Építőelemei

1. **Szekvencia:** utasítások sorba rakása
   - Tetszőleges utasítások egymás után írt csoportja.
     - Pl. értékadás, eljárás meghívása, beolvasás, kiírás
   - A szekvencia végrehajtásának **hatása:** a sorban egymás után való utasítások végrehajtásának hatása.
2. **Elágazás** (szelekció)
   - Egy programozási nyelvben akkor beszélünk elágazásról, mikor **a vezérlés valamilyen feltételtől függően más-más ágon folytatódik**.
   - Ez a vezérlési szerkezet jól használható programfutás közbeni események vizsgálatára.

# Építőelemei

**3. Ciklus** (iteráció, ismétlő struktúra)

- A ciklus, vagy iteráció az ismétlődő (azonos vagy hasonló) tevékenységek megvalósítására szolgál.
- A ciklusok három alaptípusba sorolhatók aszerint, hogy hányszor futnak le: elöltesztelő, hátultesztelő és számlálós ciklus.

# Nem a strukturált programozás részei

- A program tetszőleges helyére (címkére) ugró utasítás: **Goto**.
- A számlálós ciklus megszakítását eredményező **Break** típusú eljárások
- Eljárásból kiugró **Exit** típusú vezérlési utasítások
- A programot befejező, **Halt** típusú utasítások
- Ezek használata veszélyes, a program strukturáltságát bontják, áttekinthetősége leromlik.

# goto

- Bohm, Jacopini: a `goto` utasítás segítségével megírt bármely program megírható pusztán a strukturált programozás alapelemeinek használatával is
- A strukturált programozás, mint módszer általánosan elfogadott, de nincs vizsgálat, bizonyíték arra, hogy hatékonyabb, jobb
- A program karbantarthatóságát vizsgáló kísérletek a strukturált programozás melletti eredménnyel zárultak
- Vessey, Weber: „a strukturál programozás használhatóságát alátámasztó érvek meglehetősen gyengék” (megbízható kísérletek hiánya)

# goto

Világos írásmód és kifejezőerő

```
        ...
        ...
címke:
        ...
        ...
        if a > 0 goto címke
        ...
```

```
...
...
while a > 0 do
...
...
endwhile
...
```

- A bal oldali kódot felülről lefelé olvasva nem világos, hogy mi a `címke` szerepe, sőt több `goto` utasítás is ugorhat ugyanezen címkére.
- A `goto` több, egymástól lényegesen különböző cél megvalósítására is használható (elágazás, ciklus, ciklusból való kilépés, eljáráshívás), ezért kicsi a kifejezőereje.
- Világosabb kódot kapunk, ha a megfelelő egyedi nyelvi elemet használjuk.

- Könnyebb a programok helyességének bizonyítása (minden program-elemnek csak egy belépési, ill. kilépési pontja van).

# goto

- A `goto` is része a programozási nyelvek eszközkészletének, ha megtiltjuk a használatát, a programozó elől elvesszük a lehetőséget, hogy értelmesen felhasználja.
- Hibakezelés (a programegység belsejéből egy hibakezelő rutinra ugorhatunk, így az egységnek több kilépési pontja lesz).
- Növelhető a teljesítmény
- Néha kifejezetten természetes a `goto` használata
- (Donald Knuth: lazább megkötés a strukturált programozásra)

# Algoritmus

**Algoritmus:**

- olyan megengedett lépésekből álló módszer, utasítássorozat, részletes útmutatás, recept, amely valamely felmerült probléma megoldására alkalmas.

**Megengedett lépések ismerete**

- általában feltesszük az elemi lépésekről, hogy:
  - függetlenek, egyik sem állítható össze például néhány más lépés egymásutánjaként;
  - relevánsak, azaz mindegyik lépés legalább egyszeri végrehajtása valóban szükséges a probléma megoldásához;
  - teljes rendszert alkotnak, azaz a probléma megoldásához szükséges valamennyi elemi lépést felsoroltuk.

# Algoritmusok közös tulajdonságai

1. **Az eljárás egyértelműen leírható véges szöveggel.**
2. **Az eljárás minden lépése ténylegesen kivitelezhető.**
3. **Az eljárás minden időpontban véges sok tárat használ.**
4. **Az eljárás véges sok lépésből áll.**

Az algoritmus fogalmát gyakorlatban a következőkre korlátozzák:

- Egy (determinisztikus) algoritmus ugyanarra a bemenetre mindig ugyanazt az eredményt adja
- Minden időpontban egyértelműen adott a következő lépés.
- A nem determinisztikus algoritmusok véletlenszám-generátort alkalmaznak, így nem érvényes az első tulajdonság.

# Algoritmus-leíró eszközök

**Mondatszerű leírás.**

- informális jellegű leírás
- felsoroljuk a lépéseket, esetleg számozzuk a lépéseket
- ugrás: beleírjuk hova kell lépni
- ugrásokkal mindig meg lehet oldani a vezérlési struktúrákat (feltétel, ciklus)

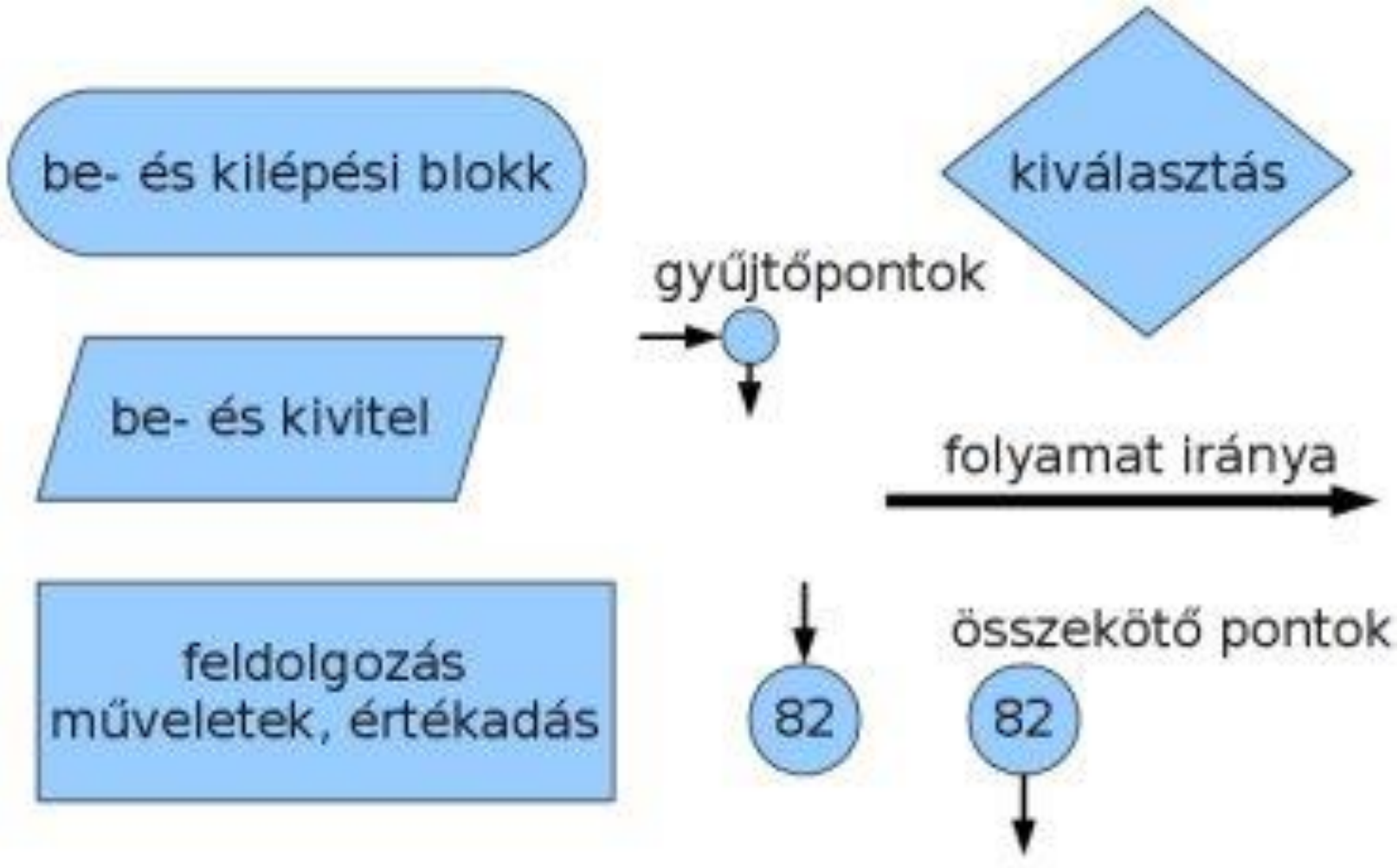
- **példa**: Eukleidész algoritmusa két szám legnagyobb közös osztójának kiszámítására (legyen a két szám: a, b)

1. felírjuk a nagyobb számot a következő alakban a = b * q + r
2. ha a felbontásban r = 0 , akkor q a legnagyobb közös osztó
3. ha r ≠ 0 akkor ( a , b ) = ( b , r ) és visszaugrunk az 1 lépésre

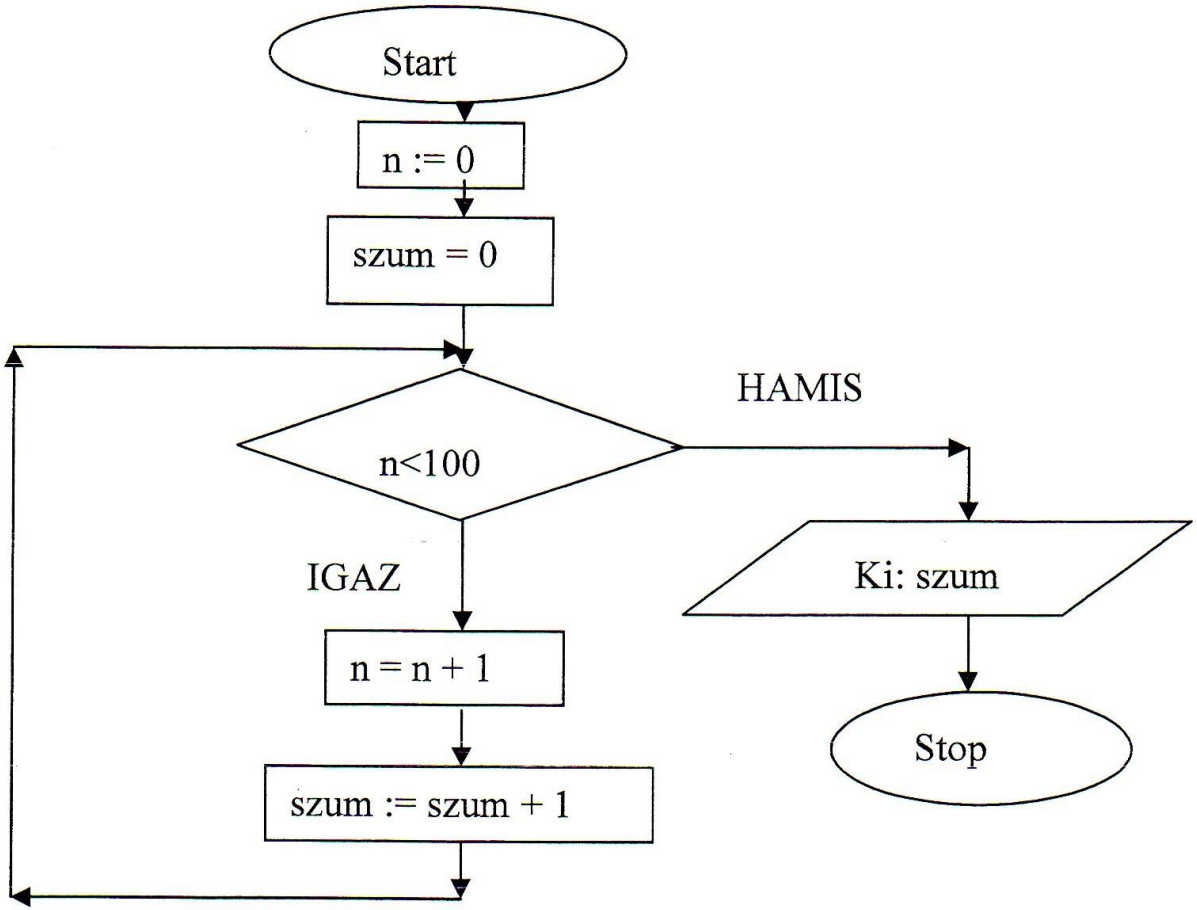
# Folyamatábra

- A folyamatábra (*flowchart*) segítségével a program dinamikus viselkedését, „folyamatát” részletekbe menően ábrázoljuk.
- Tevékenység-csomópont: téglalap
- Döntés-csomópont: az F feltételtől függően a vezérlés az igaz vagy a hamis ágon folytatódik.
- Gyűjtő-csomópont: A nyilak az algoritmus végrehajtása során összefuthatnak.
- Részletezés: A részletezéssel jelölt tevékenység egy külön folyamatábrán kerül kifejtésre.
- Program eleje és program vége
- Adatbevitel és adatkiírás
- Növekményes ciklus…

# Folyamatábra elemei

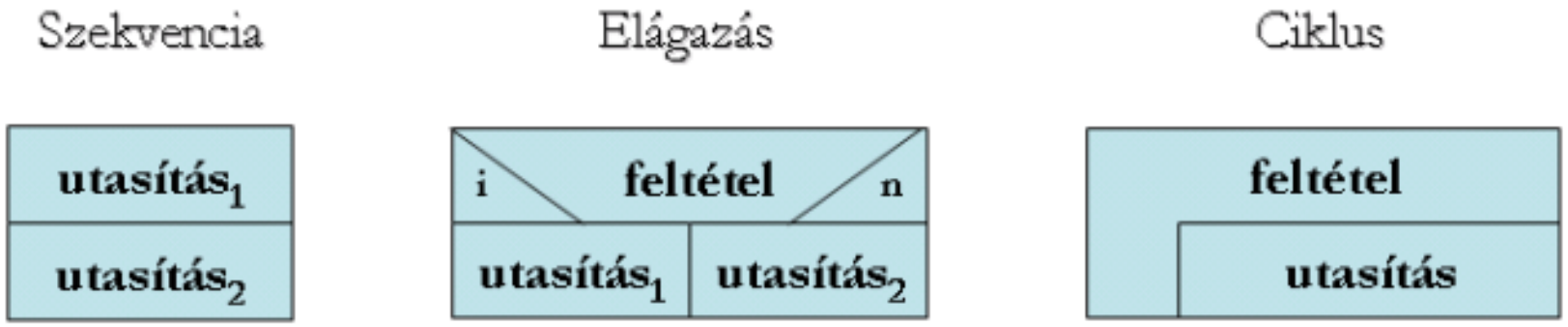


# Folyamatábra példa



# Struktogram

- Nassi-Shneiderman diagram (NSD), a strukturált programozást tükrözi, nincs lehetőség ugró utasításokra.
- A vezérlő szerkezetek a következőek:
  - szekvencia
  - elágazások
  - ciklusok



# Struktogram példa

# Pszeudokód

- „álkód”, („majdnem” kód) mivel ennek a leírási módnak egyetlen programozási nyelvvel sem azonos a formája.
- Az emberi nyelvhez közel álló, szabályokkal kötött mondatszerű leírás.
- A szabályok bizonytalanok, belekeveredhetnek programnyelvi elemek pl., a Pascal vagy a C++ nyelvek elemeiből.

# Pszeudokód

Alapvető szabályok :

1. Az algoritmus blokkszerkezetét elsősorban a tagolás tükrözi.
2. A változók neve betűvel kezdődik.
   - A változók típusát külön nem deklaráljuk,
3. A változók értékadó utasításokkal kapnak értéket.
4. Ciklusszervező utasítások: Amíg, Ismételd és Minden
5. Feltételes utasítás
6. A bevitel/kivitel a Ki és Be utasításokkal történik.
7. Eljárások, függvények hívása
8. A megjegyzések általában // jellel kezdődnek és az adott sor végéig tartanak vagy {} jelek közé íródnak.

# A feladat funkcionális felbontása

## Lépésenkénti finomítás

Ez a tervezési módszer a megírandó program által végrehajtott ***műveletekre összpontosít***.

1. *A feladat szövegének megfogalmazása*
2. *Bemeneti, kimeneti* adatok beazonosítása, esetleges adatszerkezetek kiválasztása
3. *Megszorítások* tisztázása
4. *Megoldási modellek* kiválasztása
5. Egyszerűen (***magyarul***) leírjuk azt a műveletsort, amit a programnak végre kell majd hajtania
6. Ezután a műveleteket fokozatosan ***finomítjuk***, míg el nem érjük a kellő részletezettséget - a műveletek már a használni kívánt programnyelv elemeivel is leírhatók

# Lépésenkénti finomítás

- A funkcionális felbontás módszere *Dijkstra* és *Wirth* nevéhez fűződik
- Az 1960-as években a strukturált programozással párhuzamosan fejlődött
- ***Felülről lefelé*** építkező tervezési módszer
- A funkcionális felbontás, mint tervezési módszer, elsősorban ***a program által végzett műveletekre összpontosít***, így olyan területeken alkalmazható elsősorban, ahol ezek a műveletek egyértelműek, és központi szerepet töltenek be.
- A funkcionális felbontás támaszkodik ***a tervező fantáziájára és szakmai képességeire***. Tág teret ad a kreativitásnak (ez lehet gyengeség is).

# Lépésenkénti finomítás példa

- Pattogó labdát ütögetve egy falat kell lebontanunk szöveges képernyőn
- A képernyőt egy tömbön keresztül kezeljük `ke: array[1..50,1..80,1..2]of char`
- A labda koordinátáit az `xlabda` és `ylabda` tárolja.

# Lépésenkénti finomítás példa

videojáték
  rajzold meg az oldalfalakat
  rajzold meg az ütőt a kezdőpozícióban
  `while` akar valaki játszani `do`
    rajzold meg a falat
    játszd a játékot
    jelenítsd meg a legjobb eredményt
  `endwhile`

- A konkrét részleteket itt még általános utasítások fedik el, mely lehetővé teszi, hogy tanulmányozzuk, módosítsuk a tervet anélkül, hogy az apróbb részletekkel kellene törődnünk.
- Például észrevesszük, hogy hiányzik még a legjobb eredmény kezelése

# Lépésenkénti finomítás példa

- videojáték
  rajzold meg az oldalfalakat
  rajzold meg az ütőt a kezdőpozícióban
  legjobb eredmény := 0
  `while` akar valaki játszani `do`
   eredmény := 0
   rajzold meg a falat
   játszd a játékot
   `if` eredmény > legjobb eredmény `then`
    legjobb eredmény := eredmény
   `endif`
   jelenítsd meg a legjobb eredményt
  `endwhile`

# Lépésenkénti finomítás példa

- Ha az eredmény ezen a szinten már kielégítő, nekiállhatunk az egyes lépések részletesebb kidolgozásának
- Vegyük a „játszd a játékot” eljárást:

```
játszd a játékot
    maradék labdák := 4
    while maradék labdák > 0 do
      játssz a labdával
      csökkentsd eggyel a maradék labdák számát
    endwhile
```

- Megjelent egy újabb eljárás („játssz a labdával”), melyet illik részletesebben is leírni

# Lépésenkénti finomítás példa

játssz a labdával
    rajzolj egy új labdát
    `while` a labda játékban van `do`
      vizsgáld meg a labda helyzetét
      mozgasd a labdát
      mozgasd az ütőt
    `endwhile`
    töröld a labdát a képernyőről

- Készítsük még el a ciklus három eljárásának részletezését

# Lépésenkénti finomítás példa

vizsgáld meg a labda helyzetét
    ellenőrizd, hogy játékban van-e még a labda
    ellenőrizd, hogy eltalálta-e a határfalak valamelyikét
    ellenőrizd, hogy eltalálta-e az ütőt
    ellenőrizd, hogy nekiütközött-e egy téglának

ellenőrizd, hogy eltalálta-e az határfalak valamelyikét
    `if` a labda bármelyik oldafalnál van `then`
     XLépés := - XLépés
    `endif`
    `if` a labda a felső határon van `then`
    YLépés := - YLépés
    `endif`

ellenőrizd, hogy nekiütközött-e egy téglának
    `if` ke[YLabda+YLépés, XLabda+XLépés] egy tégla helye `then`
     ke[YLabda+YLépés, XLabda+XLépés] := üres
     növeld az eredményt eggyel
     YLépés := -YLépés
     vonj le egyet a maradék téglák számából
     `if` maradék téglák száma = 0 `then` rajzolj falat `endif`
    `endif`

# Lépésenkénti finomítás példa

mozgasd a labdát
  töröld a labdát a régi helyén
  XLabda := XLabda + XLépés
  YLabda := YLabda + YLépés
  ke[YLabda, XLabda, 1] := "o„

mozgasd az ütőt
  if billentyű = "q" then mozdulj balra endif
  if billentyű = "w" then mozdulj jobbra endif

mozdulj balra
  töröld az ütőt
  if ütő helyzete > bal oldali fal + 1 then
   ütő helyzete := ütő helyzete – 1
  endif
  rajzold meg az ütőt

# Lépésenkénti finomítás értékelése

- A ***lépésenkénti finomítás*** egy folyamat, amely az algoritmus kezdetleges vázának elkészítésétől a végleges kidolgozott algoritmusig tart.
- Ahhoz, hogy a tervezés bármely szintjén megértsük a rendszer működését, nem kell pontosan ismernünk az alacsonyabb szintek működését.
- Tudnunk kell, hogy az alacsonyabb szintek mit csinálnak, de azt nem, hogy hogyan.
- Két út:
  - szintről-szintre történő finomítás (***breadth first***)
  - egy ágnak a lehető legnagyobb részletességgel való kifejtése (***depth first***)
- A pszeudokód minden részletét a strukturált programozás szabályainak betartásával írjuk le.