13

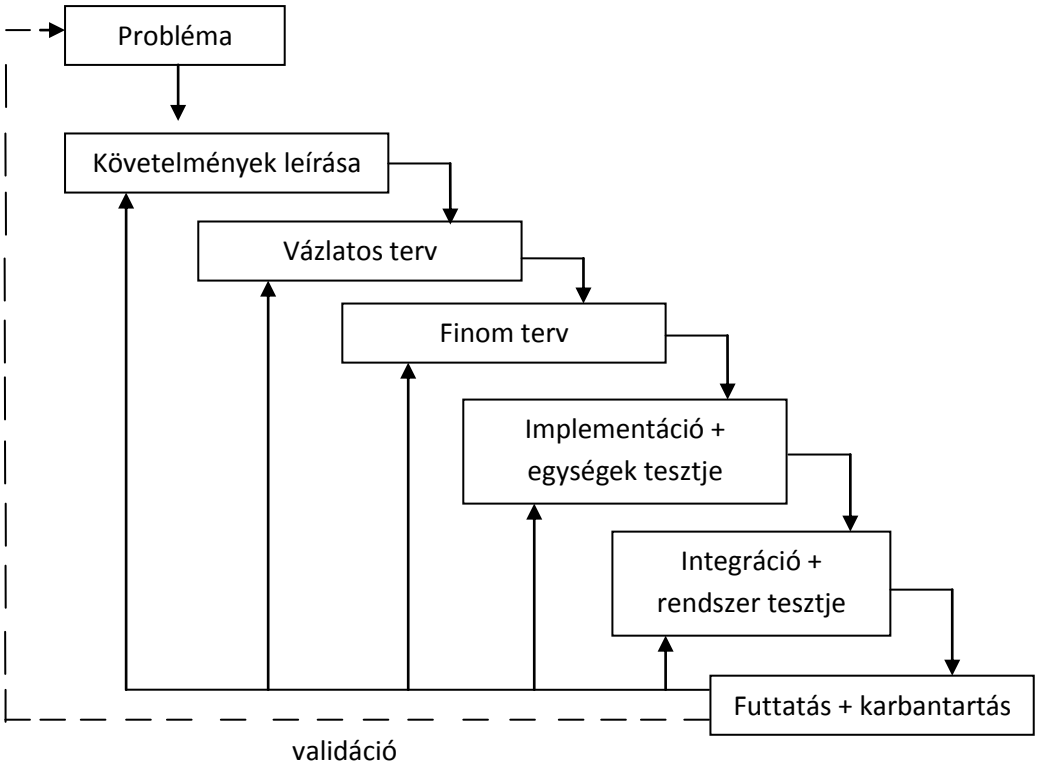
Programfejlesztési Modellek

Programfejlesztési fázisok:

- Követelmények leírása (megvalósíthatósági tanulmány, funkcionális specifikáció)
- Specifikáció elkészítése
- Tervezés (vázlatos és finom)
- Implementáció
- Verifikáció, validáció
- Rendszerkövetés, karbantartás

A fázisok közti kapcsolatok leírására többféle modellt használhatunk.

1. **Vízesés modell:** A fázisok időben követik egymást.



Hátrányai:

- nehézkes munkaszervezés
- az új szolgáltatások minden fázisra kihatnak.

2. **Evolúciós modell:**
    - A megoldás közelítő verzióinak, prototípusainak sorozatát állítjuk egymás után elő.
    - A specifikáció változhat menetközben.
    - Nehezen áttekinthető lesz a projekt.
3. **Boehm-féle spirális modell:** A program iterációs lépéseken keresztül készül el.

Iteráció 4 szakasza:

1. Célok és korlátok
2. Stratégiák, prototípusok
3. Megoldás, Validáció
4. Következő iterációs lépés megtervezése

Előnyök: Rugalmas, jól dokumentálható és tervezhető, folytonos validáció.

Hátrányok: Az emberi erőforrások elosztása nem hatékony, munkaigényes.

Objektumelvű programozás:

Procedurális: A feladatot egy leképezésnek tekintjük és a leképezési szabályt programozzuk le. (hogyan jutunk el egy előfeltételből egy utófeltételbe).

Objektumelvű programozás: A hasonló erőforrásokat használó eljárásokat egy modulba helyezzük, amelyhez szabványos hozzáférési felületet biztosítunk.

1. Adatabsztrakció: Adott szinten a megoldás szempontjából elhanyagolható részeket elhagyjuk.

2. Absztrakt adattípus konstrukciója: (A, F) pár, ahol A az adatok, F a típusműveletek halmaza.

   Osztály: (PAR, EXP, IMP, BODY) rendszer, ahol

   - PAR: A paraméterek tulajdonságainak leírása
   - EXP: Típusműveletek nevei, szintaxisa, szemantikája
   - IMP: Más osztályokból átvett szolgáltatások
   - BODY: A megvalósítás

3. Öröklődés: Egy létező superclass-ból egy új subclass-t hozunk létre származtatással.
   - A subclass átveszi a superclass tulajdonságait
   - A típushalmazok megfelelnek
   - A típushalmazok, műveletek, paraméterek átdefiniálhatóak, de jelentésük nem változhat.

   Polimorfizmus:

   - Statikus: A deklarációkor eldől.
   - Dinamikus: Az általánosabb superclass-nak adjuk értékül a subclass példányát. pl.: double x = 5; (double kap értékül int-et)

   Dinamikus összekapcsolás: Végrehajtáskor kiszámítási szabályok alapján. pl.: int x = 5.0; (automatikusan csonkol/kerekít)

Objektumelvű modellezés: A problémákat és megoldásokat különböző szempontok szerint írhatjuk le.

Statikus szempont: Milyen egységekből épül fel a rendszer?

1. Osztálydiagram: A megoldás szerkezetét leíró összefüggő gráf, melyben a csúcsokhoz az osztályokat, az élekhez pedig a köztük fennálló relációkat rendeljük. (öröklődés, asszociáció, aggregáció, kompozíció)

2. Objektumdiagram: Egyszeresen összefüggő gráf, amely csomópontjai az objektumokat, élei pedig a köztük lévő összekapcsolást jelentik (ami nyilván nem lehet öröklődés). Az objektumdiagram időben változik, de mindig létezik hozzá osztálydiagram.

Dinamikus szempont:

1. Állapotdiagram: Összefüggő irányított gráf, amelynek csúcsaihoz a program kezdőállapotából elérhető állapotokat feleltetünk meg, éleinek meg az állapotátmeneteket. Két csúcs között több állapotátmenetet is jelölhetünk, hiszen több esemény hatására is létrejöhet.

2. Szekvencia diagram: Az objektumok közötti üzenetváltások időbeli menetét szemlélteti.

3. Együttműködési diagram: Az osztályok objektumai hogyan működnek együtt a problémamegoldásban, milyen üzenetváltások vannak. Csak azok az objektumok relevánsak, amelyek osztályait az osztálydiagramban asszociációs kapcsolat köt össze.

4. Aktivációs diagram: A párhuzamos vezérlési feladatokkal együtt mutatja a problémamegoldás lépéseit.

Implementáció szerinti:

1. Komponens diagram: A problémamegoldásban résztvevő egységek (komponensek) és a köztük fennálló reláció szerepel benne. A komponensek akár kis is cserélhetőek, ha biztosítjuk a csatlakozási felületet a környezethez.
   Pl.: A véletlenszám-generátor kicserélhető más algoritmusúra, ha ugyanúgy paraméterezhető.

2. Alrendszer diagram: Modulok, csomagok hierarchiája és együttműködésük.

Környezet szerinti: Hardware és software igény.

Használati eset szerinti: A használati eset diagram a felhasználók szempontjából mutatja meg a programszolgáltatások működését. Ez teremthet kapcsolatot a megrendelő és a projektvezető elképzelései között.

Tervminták: Az objektumelvű rendszertervek tervezése nehéz. A probléma megoldása mellett a terv újrafelhasználhatósága is cél. A tervminták az újrafelhasználható tervek és tervrészek.

4 alapvető elemből épül fel:

1. Név
2. Feladat leírása: Célok és motiváció
3. Megoldás: Szerkezet, komponensek, együttműködés, implementáció (akár példakód)
4. Következmények: Hatékonyság, hasonló problémák