1. Bevezetés

1.1. A szoftverrel szemben támasztott követelmények felhasználói oldalról

- feltétlen megbízhatóság, üzembiztosság (reliable)
- könnyű karbantarthatóság, nyomonkövethetőség (maintainable)
- hatékony működés (efficient)
- felhasználóbarát felület (user-friendly)
- egyszerű továbbfejleszthetőség
- futási idő minimalizálás
- tárigény minimalizálás
- gyors és olcsó kivitelezés
- határidők pontos betartása
- egyéniség független programozás
- gépfüggetlen szoftver
- jól dokumentált szoftver

1.2. A szoftverfejlesztő számára fontos szempontok:

- piac által elfogadott
- minél több újrafelhasználható komponense legyen
- hírnév
- haszon
- új termékvonal
- tapasztalat
- új készségek a team-nek
- módszertan, team-fejlődése

1.3. A szoftver krízis (1967 NATO-konferencia: szoftverválság)

1.3.1. A szoftverkrízis tünetei

- a programok megbízhatatlanok (pl. nem tudja a specifikációt)
- a programok alkalmazkodásra képtelenek (operációs rendszer változásra érzékeny, konfiguráció változásakor körülményes az átparaméterezés)
- nehézkesek (nem állíthatók össze részprogramokból, bővítés aránytalanul nagy munkával jár)
- udvariatlanok (helytelen beavatkozás nem javítható, a program lefagy, csak azt írja ki, hogy gond van, de hogy mi, azt nem)

1.3.2. A krízis okai

- az előállított szoftverek méretének növekedése maga után vonta a komplexitás növekedését
- minőségi követelmények változása
- fejlesztési módszerek nem tartottak lépést a változással
- felhasználói környezet változása

1.3.3. Megoldás

- a szoftverkészítés technologizálása
- új elvek, módszerek kidolgozása
- szoftver szabványok bevezetése
- új programozási paradigmák alkalmazása

1.4. Szoftverfejlesztés lépései:

1. Elemzés (Requirements Analysis)
2. Specifikáció (Specification)
3. Rendszer és szoftver tervezés (System and software design)
4. Implementáció (Implementation)
5. Tesztelés (Verification, Validation, Testing)
6. Üzemeltetés, karbantartás (Operation and Maintenance)

1.5. Szoftverfejlesztés modelljei:

1.5.1. Vízesés modell (waterfall model)

A szoftverfejlesztés lépcsőin sorban végigmegy, és a következő lépcsőre akkor lép, ha az előző feladatot elvégezte.

Jellemzők:

- hagyományos mérnöki szemléletet követ
- leginkább elterjedt, legrégibb modell

Problémák:

- a valós projektek ritkán követnek szekvenciális modellt
- nehezen valósítható meg az iteráció
- egész modell a specifikáció minőségétől függ
- a projekt elején meglévő kezdeti bizonytalanságot nem tudja kezelni
- nagyon későn lát a megrendelő működő programot
- nincs tapasztalat a fejlesztés közben
- nem támogatja az újrafelhasználhatóságot

Előnyös ha:

- egyszerű, érthető a feladat,
- jól definiálható a környezet
- kevés újrafelhasználható komponens látszik

1.5.2. Gyors modellezés (prototyping)

A felhasználó nem definiálja pontosan a projekt elején a követelményeket, illetve fejlesztés közben módosul a specifikáció. Gyorsan elkészítenek egy előzetes verziót (intuíciót felhasználva), majd a későbbiek folyamán ezt fejlesztik tovább a követelményeknek megfelelően. Újabb és újabb prototípusok gyártásával juthatunk el egy minőségi rendszerhez.

Előnyök:

- gyorsan elkészül a 0. változat
- könnyebb dönteni a folytatásról
- pontosabb, teljesebb prototípusok készülnek

Problémák:

- pazarlás, ha az előzetes verziót eldobjuk

- a megrendelő nem hagyja élni a fejlesztőt
- megvan az esélye annak, hogy egy termékbe prototípus szintű megoldások is kerülnek

Előnyös:

- ha gyorsan kell elkészíteni egy használható verziót
- amikor a felhasználói igények nincsenek pontosan definiálnva ('nehéz felhasználó' látszik)
- inkább az alkalmasság mint a pontosság az ami a tervezők célja
- változnak a feltételek
- kísérleti fejlesztésekről van szó (lehet hogy nem kell a termék)
- ha viszonylag kis termékről van szó
- AI (artifical intelligence) rendszerek fejlesztésénél

1.5.3. Programnövesztés

A feladat kis részekre bontjuk, és azokat precízen kidolgozzuk, majd a részeket összekapcsoljuk.

Hátrányok:

- termék minden része csak egyszer fut át a programfejlesztés hagyományos fázisain
- rossz döntés könnyen bekerülhet

Előnyös, ha:

- költség vagy határidő nyomás erős
- rögzített, jól definiált igények vannak
- hiányzó önbizalom a programozó részéről
- bizalom hiánya a vevő oldalról

1.5.4. Újrafelhasználhatóság (system assembly from reusable components)

Egy rendszert már létező, előzetesen definiált komponensekből építjük fel.

Előnyök:

- ez a leggazdaságosabb út
- felhasznált komponensek megbízhatóbbak lesznek, mert többfajta környezetben voltak tesztelve
- egy-egy komponens megbízható, mert többszöri újraírás eredménye

Hátrányok:

- komponensek illesztése nehézkes lehet
- módosítás nehézkes lehet

Előnyös, ha :

- költség és határidő nyomás esetén
- ha az adott területen vannak felhasználható komponensek

1.5.5. Magassztintű nyelvekkel, vagy alkalmazásgenerátorokkal (formal transformation)

Előnyök:

- csak a problémát kell definiálni a neki megfelelő nyelven
- a kódolás utána gyorsan megy, lényegében már csak a teszteléssel van gondunk

- gyors, és a hibamentesség szempontjából jó kódot ír

Hátrányok:

- nem biztos, hogy hatékony a megoldás
- nagy programok születnek

Előnyös, ha :

- ha szabványos alkalmazási területen dolgoznak