**Fiatal Muszakiak Tudományos Ülésszaka,
Kolozsvár, 1998. márc. 20-21., pp. 97-99.**

SZOFTVEREK MINOSÉGBIZTOSÍTÁSA PARADIGMÁK HASZNÁLATÁVAL

Hornyák Olivér

This paper focuses on quality assurance of software products. It describes the standards concerned such as ISO 9000-3 and the Capability Maturity Model.
The second part deals with the software crisis and how developers overcome and summarise the well-known methods and paradigms of software engineering and their properties.

Minoségbiztosítási szabványok

Az információs technológia fejlodésével, a szoftvertermékek számának növekedésével a szoftverek minoségbiztosítása nagyon lényegessé vált. Az ISO 9000 sorozatú szabványok arra késztetik a különbözo termékek eloállítóit, hogy ezen szabványoknak minél jobban megfeleljenek. Nem kivétel ezalól a szoftver mint termék sem, bár sajátos jellegénél fogva a szoftver minoségének megragadása, számszerusítése és mérése nem könnyu feladat.
A szoftver minoségének kritériumait vizsgálhatjuk mind a felhasználó, mind a fejleszto szempontjából. Ezek a kritériumok a következok:

- helyesség
- felhasználó-barátság
- robosztusság
- kompatibilitás
- hatékonyság
- szabványosság
- karbantarthatóság
- bovíthetoség
- újrafelhasználhatóság
- hordozhatóság
- ellenorizhetoség.

A szoftvertermékek minoségbiztosításának tekintetében Európában az ISO 9000-3 ajánlására hivatkoznak [1], míg az Egyesült Államokban a „Capability Maturity Model”, CMM a leginkább elfogadott.
Az ISO 9000-3 irányelveket ad, melyek megadják:

- az alapdefiníciókat (szoftver, szoftver termék, fejlesztés, fejlesztési fázis, ellenorzés)

- a kereteket (szervezeti formák, felelosségi körök)
- életciklus tevékenységeket (specifikáció, analízis, tervezés, implementáció, tesztelés, karbantartás)
- kiegészíto tevékenységeket (konfiguráció, dokumentáció, betanítás).

A CMM modell kiértékelése alapján egy adott munkafolyamat a következo öt kategória valamelyikébe sorolható:

- kezdetleges
- megismételheto
- jól meghatározott
- szervezett
- optimalizált

A fenti szabványok nem kötik meg az alkalmazható fejlesztési módszertanokat. Rögzítik viszont, hogy kelloen definiált, kezelheto módszertanokat kell alkalmazni.

Szoftverfejlesztési paradigmák

A szoftverfejlesztés kezdeteit a strukturálatlanság jellemezte. Az általában assembly nyelven írt kódok nélkülözték az átfogó tervezési meggondolásokat. Ezen a helyzeten az elso magas szintu programozási nyelvek megjelenése nem sokat változtatott. A vezeto programozó kiválása gyakran a projekt kudarcát is okozhatta. Senki sem tudta megjósolni, meddig tart a rendszer kifejlesztése, milyen komplex lesz a rendszer, milyen méretu lesz a rendszer, mire elkészül. Nehéz volt megmondani, hogy a szoftver életciklusában hol járunk. A kialakult jelenséget szoftver krízisnek nevezzük.

A krízisbol a strukturált programozás módszerének megjelenése vezetett ki. A strukturált más néven funkcionális stratégiák szerint a programok funkcionális szempontból tervezendok. A munka felülrol indul, és a részletek feltárásával halad lefelé. A rendszer állapota oszthatatlan egész, egyetlen helyen van számontartva, és ehhez a helyhez (állapotleíráshoz, adatbázishoz) minden funkció hozzáférhet.

A strukturáltság szigorú betartásának a hatékonyság csökkenése lehet az eredménye. Elofordulhat, hogy egy magasabb szintu rétegben olyan egyszeru muveletekre is szükség van, amelyekre az alacsonyabb rétegekben is számítanak. Ha ilyenkor következetesen betartjuk a strukturált programozás elveit, akkor több rétegen keresztül is csupán közvetíto szerepet betölto funkciók jelenhetnek meg. A rendszer állapotváltozóihoz való közvetlen hozzáférés rendkívüli veszélyeket rejt, és nehezen felfedezheto hibák forrásává válhat.

A programok méretének növekedésével egyre fontosabbá vált a dekompozíció, azaz a feladat külön kezelheto és önállóan fordítható és tesztelheto részekre való bontásának igénye. Ezt a szoftvertechnológiai áramlatot moduláris programozásnak nevezzük. A moduláris programozást a modulon belüli eros összetartás és a modulok közötti laza kapcsolat jellemzi.

A szoftverek fejlesztésénél modelleket állítunk fel, melyek a valós világ objektumait képezik le az informatika síkjára. Ez vezetett az objektumorientált paradigma megjelenéséhez. Paradigmának nevezünk egy nagyobb komplex rendszer szemléletét, látás és gondolkodásmódját, amely az adott fogalomkörben használt elméletek, modellek, és módszerek összessége jellemez. [2]

Az objektum olyan modellje a világ egy részének, amely a számára kívülrol érkezett üzenetekre reagálva valahogyan viselkedik. Az objektumnak kívülrol nem látható belso statikus struktúrája van, amely az állapotok értékét rögzíto attribútumokkal definiálható. Beszélhetünk az objektum állapotáról, amely a belso struktúrát egy adott pillanatban kitölto értékek halmaza. A megegyezo viselkedésu és struktúrájú objektumok egy közös minta alapján készülnek, amit osztálynak nevezünk. Az objektum tehát az ot definiáló osztály egy példánya.

Az objektum-orientált tervezés az analízisnél kezdodik. Az analízis során a feladat definíciójából az alapveto objektumok, és a rájuk ruházott felelosségek összegyujthetok. Az analízis során eloállíthatóak absztrakt modellek, amelyeket le kell képeznünk a fizikai rendszerünk által nyújtott szolgáltatásokra. Ezt a leképzést nevezzük tervezésnek. Az objektumorientált szemlélettel tervezett szoftverek minoségének kulcsa az objektumok zártságában rejlik. Az egyes objektumok csak üzeneteken keresztül tudnak kommunikálni egymással, minden objektumnak saját állapota van, és ezekhez az állapotváltozókhoz nincs közvetlen, külso hozzáférés. A már megírt objektumok újrafelhasználhatósága megoldott, a fejlesztorendszerek gyártói elore megírt objektumrendszerekkel segítik a hibátlan szoftverek eloállítását.

Az objektumorientált szemlélet jól megfigyelheto a szimulátoroknál. A szimulátorok a valós világ objektumait modellezik úgy, hogy azoknak csak a modellezés szempontjából fontosnak tartott tulajdonságait tartják meg. Ezek az objektumok az oket ért hatásokra valamilyen módon viselkednek, és ez minél jobban hasonlít a valós objektum viselkedésére, annál tökéletesebb a szimuláció. A szimulátorok esetét általánosítva azt is mondhatjuk, hogy az objektumorientáltság nem csupán szoftvertechnológiai paradigma, terjedése a muszaki élet egyéb területein is megfigyelheto.

Irodalomjegyzék

[1] DIN ISO 9000 Teil 3 1991, pp 4-31

[2] Dr. Kondorosi Károly - Dr. László Zoltán - Dr. Szirmai-Kalos László: Objektum-orientált szoftverfejlesztés; ComputerBooks Budapest ,1997, pp. 33-63

[3] Angster Erzsébet : Az objektumorientált tervezés és programozás alapjai; Martonvásár, 1997, pp.1-30

Hornyák Olivér

Miskolci Egyetem , Alkalmazott Informatika Tanszék ,
3515 Miskolc-Egyetemváros
36-46-365-111/19-07