FIATAL MŰSZAKIAK TUDOMÁNYOS ÜLÉSSZAKA

Kolozsvár, 1998. március 20-21.

Szoftverek minőségbiztosítása paradigmák használatával

Hornýák 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.

Minőségbiztosítási szabványok

Az információs technológia fejlődésével, a szoftvertermékek számának növekedésével a szoftverek minőségbiztosítása nagyon lényegessé vált. Az ISO 9000 sorozatú szabványok arra készítetik a különböző termékek előállítóit, hogy ezen szabványoknak minél jobban megfeleljenek. Nem kivétel ez alól a szoftver mint termék sem, bár sajátos jellegénél fogva a szoftver minőségének megragadása, számszerűsítése és mérése nem könnyű feladat.

A szoftver minőségének kritériumait vizsgálhatjuk mind a felhasználó, mind a fejlesztő szempontjából.

Ezek a kritériumok a következők:

- helyesség	- kompatibilitás	- bővíthetőség
- felhasználó-barátság	- hatékonyság	- újrafelhasználhatóság
- robosztusság	- szabványosság	- hordozhatóság
	- karbantarthatóság	- ellenőrizhetőség.

A szoftvertermékek minőségbiztosításának 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, ellenőrzés)
- a kereteket (szervezeti formák, felelősségi körök)
- életciklus tevékenységeket (specifikáció, analízis, tervezés, implementáció, tesztelés, karbantartás)

- kiegészítő tevékenységeket (konfiguráció, dokumentáció, betanítás).

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

- kezdetleges
- megismételhető
- jól meghatározott
- szervezett
- optimalizált

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

Szoftverfejlesztési paradigmák

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

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

A strukturáltság szigorú betartásának a hatékonyság csökkenése lehet az eredménye. Előfordulhat, hogy egy magasabb szintű rétegben olyan egyszerű műveletekre is szükség van, amelyekre az alacsonyabb rétegekben is számítanak. Ha ilyenkor következetesen betartjuk a strukturált programozás elveit, akkor több rétegen keresztül is csupán közvetítő szerepet betöltő funkciók jelenhetnek meg. A rendszer állapotváltozóihoz való közvetlen hozzáférés rendkívüli veszélyeket rejt, és nehezen felfedezhető hibák forrásává válhat.

A programok méretének növekedésével egyre fontosabbá vált a dekompozíció, azaz a feladat külön kezelhető és önállóan fordítható és tesztelhető részekre való bontásának igénye. Ezt a szoftvertechnológiai áramlatot moduláris programozásnak nevezzük. A moduláris programozást a modulon belüli erős 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 kívülről érkezett üzenetekre reagálva valahogyan viselkedik. Az objektumnak kívülről nem látható belső statikus struktúrája van, amely az állapotok értékét rögzítő attribútumokkal definiálható. Beszélhetünk az objektum állapotáról, amely a belső struktúrát egy adott pillanatban kitöltő értékek halmaza. A megegyező viselkedésű és struktúrájú objektumok egy közös minta alapján készülnek, amit osztálynak nevezünk. Az objektum tehát az őt definiáló osztály egy példánya.

Az objektum-orientált tervezés az analízisnél kezdődik. Az analízis során a feladat definíciójából az alapvető objektumok, és a rájuk ruházott felelősségek összegyűjthetők. Az analízis során előállíthatóak absztrakt modellek, amelyeket le kell képeznünk a fizikai rendszerünk által nyújtott szolgáltatásokra. Ezt a leképzést nevezzük tervezésnek. Az objektumorientált szemlélettel tervezett szoftverek minőségének kulcsa az objektumok zártágában rejlik. Az egyes objektumok csak üzenetekre keresztül tudnak kommunikálni egymással, minden objektumnak saját állapota van, és ezekhez az állapotváltozókhoz nincs közvetlen, külső hozzáférés. A már megírt objektumok újrafelhasználhatósága megoldott, a fejlesztőrendszerek gyártói előre megírt objektumrendszerekkel segítik a hibátlan szoftverek előállítását.

Az objektumorientált szemlélet jól megfigyelhető 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 őket 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 műszaki élet egyéb területein is megfigyelhető.

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