Miskolci Egyetem
Alkalmazott Informatikai Intézeti Tanszék
A minőségbiztosítás informatikája

SZOFTVER-MINŐSÉGBIZTOSÍTÁS

Készítette: Urbán Norbert

Szoftver-minőség

- A szoftver egy termelő-folyamat végterméke,
- A minőség azt jelenti, hogy a fejlesztési folyamat a specifikációs követelményeket kielégíti.

Szoftver-minőség

Szoftverek esetében nehézségek:

- A specifikációnak a felhasználó által kívánt termék karakterisztikáját kell követni,
- A specifikációban vannak nem szereplő követelmények is(fejlesztési szervezet),
- A szoftver specifikációk rendszerint nem teljesek.

Szoftver-minőség menedzserek tevékenységei

- Minőségbiztosítás
- Minőségtervezés
- Minőségszabályozás

Szoftver minőség szabványok

A minőséggel kapcsolatos szabványok:

- ISO 9003 (termék),
- ISO 9002 (termék, gyártás),
- ISO 9001 (tervezés, gyártás, értékesítés),
- ISO/IEC 12207 (szoftver életciklus folyamatok),
- ISO 9126 (szoftver minőségi karakterisztikák),
- ISO/IEC 14598 (szoftver termék),
- ISO 9127 (felhasználói dokumentáció),
- IEEE Std. 830 (követelmény specifikáció).

Szoftver minőségi karakterisztikák

- Funkcionalitás(alkalmasság, pontosság, védelem)
- Megbízhatóság(hibatűrés, visszaállíthatóság)
- Használhatóság(érthetőség, működtetés)
- Hatékonyság(idő és erőforrásbeli)
- Karbantarthatóság(elemezhetőség és tesztelhetőség)
- Hordozhatóság(adaptálhatóság, cserélhetőség)

Biztonságkritikus rendszerek

Olyan informatikai rendszer, amely azzal az elsődleges követelménnyel működtetendő, hogy ne veszélyeztessen semmit, ne okozzon károkat (biztonságos működés).

Másik fontos tulajdonság a megbízhatóság.

- Hibatűrő rendszer.

Biztonságkritikus rendszerek

Minőségi és megbízhatósági követelmények:

- Megbízhatóság
- Rendelkezésre állás
- Biztonságosság
- Teljesítőképesség
- Karbantarthatóság

Biztonságkritikus rendszerek

A hibatűrés megvalósítása előre betervezett redundanciával kell, hogy járjon.

Szoftver redundancia: a hibatűrés elérése érdekében kiegészítő szoftverelemeket, szoftverrel megvalósított módszereket alkalmaznak.

Biztonságkritikus rendszer - A szoftver redundancia megvalósítása

A szoftver hibatűrés két céllal valósítható meg:

- Szoftver saját hibájának elfedésére,
- Szoftver saját és hardver hibáinak elfedésére.

A szoftver hibatűrést szolgáló megoldások:

- N-verziós programozás,
- A javító blokkok módszere.

Szoftver hibák

a) Szoftver specifikációs hibák

A fejlesztés kezdetekor megjelenő hibák, amelyek a szoftver előre megadott, specifikált működési funkcióiban nyilvánulnak meg, mégpedig oly módon, hogy a szoftver valamilyen vonatkozásban nem teljesíti a felhasználói követelményeket. A specifikációs hiba adódhat: téves, ellentmondásos, valamint hiányos specifikációból.

Szoftver hibák

b) Programozói hibák

A szoftver tervezése és kódolása során a programozó által elkövetett hibák körét foglalja magába. Lehetséges hibák:

- hiányzó funkciók,
- indítási és leállítási hibák,
- felhasználói interfész hibái,
- kódolási hiba,
- algoritmus hiba,
- stb.

Szoftver-tesztelési módszerek

Tesztelési elvek:

- teszt szükséges része az elvárt kimenet vagy eredmény definiálása,
- célszerű a tesztelést a fejlesztéstől független személy vagy szervezet által elvégeztetni,
- a teszteket nem csak elvárt feltételekre kell írni, hanem olyanokra is, amiket nem várunk el,
- a tesztelés igen kreatív tevékenység, komoly szellemi kihívást jelentő feladat.

Szoftver-tesztelési módszerek – Funkcionális tesztelés

- Ekvivalencia partícionálás
- Határérték-analízis
- Ok-hatás-analízis
- A véletlenszerű tesztgenerálás felhasználása

Szoftver-tesztelési módszerek – Strukturális tesztelés

- Hibamodell ST esetén
- A tesztelés menete ST esetén
- A strukturális tesztelés előnyei és felhasználása
- A vezérlési szerkezet modellezése
- Programok vezérlési bonyolultsága
- Egyszerű strukturális tesztgenerálási algoritmus
- ST általános problémája: Ciklusok tesztelése

Tesztelési stratégiák és folyamatok

Szoftverfejlesztési modellek:

- vízesés modell,
- spirál modell,
- V-modell.

Verifikáció és validáció.

Egységek, modulok tesztelése.

A tesztelés költsége.

Bizonylatolás.

Objektum orientált szoftvertesztelés

Az OO típusú problémamegoldásakor problémát okozható jellemzők:

- Problémák szétdarabolása,
- Adatrejtés.

Tesztelés lehetséges nem OO rendszerekben:

- top-down tesztelés.

Objektum orientált szoftvertesztelés – Tesztelés nem OO rendszerekben

Izolációs tesztelés:

- szoftver komponensei egyenként tesztelhetőek,
- a belső működése tesztelhető először a komponenseknek, majd az interfészek,
- a tesztkörnyezet adott szoftver verzióiban újrafelhasználható,
- a tesztelés megfelelő teszt környezet esetén jól automatizálható.

Objektum orientált szoftvertesztelés – Hagyományos tesztelési módszerek

Izolációs tesztelés:

- a környezet kialakítása okozza legnagyobb gondot (szoros kapcsolat komponenseknél),
- probléma a teszt környezet újrafelhasználása,
- nem gyakran alkalmazzák OO tesztelések esetén.

Objektum orientált szoftvertesztelés – Hagyományos tesztelési módszerek

Bottom-up tesztelés:

- a tesztelő részekre osztja a SUT-ot,
- legelőször azon komponensek tesztelése, melyek nem függenek más osztálytól,
- ezt követően azok az osztályok következnek, melyek az először tesztelt osztálytól függenek,
- tesztelt osztályra épülő osztályok tesztelése addig tart, míg a szoftvert le nem teszteltük.

Objektum orientált szoftvertesztelés – Hagyományos tesztelési módszerek

Bottom-up tesztelés:

- legnagyobb problémát a körkörös hivatkozások jelentik,
- hátránya még az adott rétegben lévő komponensek ugyanazon tesztesetekkel teszteljük,
- nem tesztelhető egy adott réteg viselkedése, ha az alatta lévő réteg hibásan működik.

Konkrét tesztelő rendszerek

- Cantata++
- SGI Tester
- Aprobe

Köszönöm a figyelmet!