Schuster György, Terpecz Gábor, Tokodi Dániel

SZOFTVER MEGBÍZHATÓSÁG

A szoftver megbízhatóság egy adott, meghatározott időtartamon belül, adott környezetben a kérdéses szoftver hibamentes működésének valószínűsége. A szoftver kritikus sikertényező és mint ilyen nem kerülhető ki. Ennek a következménye az, hogy a szoftver megbízhatósága egyértelműen kihat a rendszer teljes megbízhatóságára. További tendencia az, hogy a rendszerekben szoftverek bonyolultsága egyre növekszik azért, hogy a rendszer, a környezet és a felhasználó hibáit kiküszöbölje. Jól ismert tény az is, hogy minél összetettebb egy rendszer annál nagyobb a hiba lehetősége. A megbízhatóság vizsgálatának számos jól bevált módszere van a hagyományos rendszerek esetén. Például számítható az anyagfáradás, a kopás, vagy akár a kenőanyagok tulajdonságainak megváltozása. A szoftver megbízhatóság becslésére és vizsgálatára ezek a módszerek nem alkalmasak. Míg a hagyományos rendszereknél az üzemeltetés körülményei és a karbantartás nagyban befolyásolják a megbízhatóságot, addig szoftverek esetén lényeges hangsúly van a tervezés helyességén.

Kulcsszavak: *szoftver, megbízhatóság, rendszer, biztonságkritikus, tesztelés*

BEVEZETŐ

Esetek:

- 2015. május 9-én egy Airbus A400M Atlas szállító-repülőgépgép berepülési feladata közben Seville közelében földnek ütközött és megsemmisült, fedélzeten tartózkodó hat emberből négyen életüket vesztették, ketten súlyosan megsérült. A gép felszállás után röviddel a négy hajtóművéből háromban működés problémák léptek fel. A hiba oka szoftver problémára vezethető vissza [6].
- 2015. május 1-jén az FAA[1] kiadott egy figyelmeztetést és egy utasítást egy szoftver hiba kijavítására, amely a teljes elektronikus rendszer kiesését okozhatja Boeing 787 Dreamliner-ek esetén. A hibát a repülőgép generátor vezérlőjében találták [7].

Átlagos megfigyelő azt gondolhatja, ha egy szoftver egyszer már hibátlanul futott, akkor az többé nem hibázik. A valóság ettől eltérhet.

Mivel a szoftverek – hasonlóan az emberi irányítókhoz – döntéseket hoznak a különböző környezeti hatások ezeket meg is zavarhatják.

Korábban már hibátlanul futó vezérlő program, amely az Ariane 4 rakétán probléma nélkül futott az Ariane 5 első repülésekor a vezérlés elvesztéséhez, végül annak megsemmisüléséhez vezetett. Az ok egyszerű: egy változó túlcsordult[2] [8].

[1] FAA – Federal Aviation Administration , Szövetségi Légügyi Hivatal

[2] túlcsordulás: Számábrázolási tartomány túllépés

Definíciók

Szoftver: egy adott számítógépre, mikrovezérlőre megírt programok és az adatok összessége[3].

Szoftver megbízhatóság: A szoftver megbízhatóság egy adott, meghatározott időtartamon belül, adott környezetben a kérdéses szoftver hibamentes működésének valószínűsége [9].

Eltérés (failure): a felhasználó által észlelt váratlan szoftver viselkedés.

Hiba (fault): az eltérés által okozott szoftver jellemző.

Befolyásoló körülmények

A szoftverek megbízhatóságát az alábbi tényezők befolyásolják [9]:

- emberi tényező. A programok és a szükséges adatok előállítóinak tapasztaltsága és képessége a hibamentes munkára. Hasonlóan szükséges a tesztelést végző személyzet képzettsége.
- az alkalmazott fejlesztési modell. Alapvetően a mostanság divatos fejlesztési alapelvek agile[4] és scrum[5] biztonságkritikus fejlesztésekben kerülendők.
- a menedzsment alkalmassága. Sokszor felmerülő probléma az alkalmatlan menedzselés. A menedzser típusú vezetők sok esetben nem értenek az adott szakterülethez – tisztelet a kivételnek – és olyan dolgokat várnak el a fejlesztőktől, amely, vagy értelmetlen, vagy teljesíthetetlen.
- az alkalmazott fejlesztői környezet hibamentessége. Számos esetben előfordul, hogy egy hibátlan forráskód a fordító program, vagy az alkalmazott programkönyvtárak hibája miatt nem működik megfelelően.
- a tesztelő környezet hibamentessége. A tesztelés lényeges része a szoftver életciklusának, így megbízhatósági szempontból lényeges kérdés. A tesztek nagy részét szükséges automatizálni, főleg, ha az állóképességi és időtartam teszt. Amennyiben a tesztelési környezet hibás, vagy hiányos egyértelműen belátható, hogy az eredményt befolyásolja.
- a hardver környezet kiválasztása. A szoftver hardver nélkül semmit nem ér. Ha a hardver kellő platformot nyújt az elvégzendő feladatokhoz elegendő tartalékkal, akkor van esély a szoftver hibátlan futására. Előfordult már olyan eset, hogy egy tesztelt hardver – szoftver konfiguráció a kiválasztott szimulációs környezetben és a valóságban több évig jól működött, de egyszer csak kifogyott a rendszer hardver a tartalékaiból és a rendszer leállt, illetve hibásan működött.

Szoftver megbízhatóság mérőszámai

A szoftver megbízhatóságot befolyásoló tényezőkre mérőszámokat kellene meghatározni. Azonban míg más mérnöki területeken ezek kiválasztása viszonylag egyszerű és egyértelmű, addig ezen a területen az objektív befolyásoló tényezők megtalálása igen bizonytalan [1].

[3] A működő rendszer mindazon része, amely nem anyagiasítható (nem fogható meg).

[4] agile: „agilis” fejlesztési modell

[5] scrum: scrum fejlesztési modell

A legfontosabb befolyásoló tényezők

A szoftver komplexitása

Egyszerű mérőszámnak tűnik, de nem az. A kérdés az, hogy milyen mutató alapján próbáljuk ezt a jellemzőt meghatározni. Ez lehet:

1. a program kódsorainak száma. Valóban alkalmazott mérőszám a LOC[6] , illetve a KLOC[7]. Ez az adat kiindulásnak jó, de nem igazán mérvadó. Például: ha egy viszonylag hosszú program egyetlen szálként fut, annak komplexitása nyilvánvalóan nem éri el azon program komplexitását, amely esetleg több szálon fut.
2. a szoftver rendszer fájljainak száma. Tapasztalt programozók sok esetben pont a túlzott komplexitás elkerülése miatt a működő rendszert több működő processzre szedik szét azért, hogy uralják a bonyolultságot és particionáltan tesztelhetővé tegyék a különböző részek tesztelését.
3. a párhuzamosan futó feladatok száma. Ez egy elég jó mérőszám lehet, de ez sem elegendő egyedül.
4. az alkalmazott IPC[8] elemek száma. Ezek a LOC mérőszámmal együtt már elég jó becslést nyújtanak, de közel sem teljeset. Az IPC elemek száma viszont utalhat a különböző speciális meghibásodások előfordulásának valószínűségére (Dead Lock).

A programozók tapasztaltsága

Szinte megoldhatatlan és mérhetetlen mérőszám. A szoftverek nagy része nem is egy programozó, sőt nem is azonos csoport tevékenységének eredménye. Így ez az egyébként igen fontos mérőszám még csak nem becsülhető.

A projekt menedzsment mérőszámai

Közismert tény, hogy egy jól irányított projekt nagyobb valószínűséggel eredményez jó eredményt. Amennyiben a projekt menedzselése támogató és normálisan értékelő, akkor a fejlesztői munka sokkal nyugodtabb és eredményesebb, mint például egy folyamatos értelmetlen számon kérő esetben. A probléma továbbra is az, hogy erre konkrét mérőszámot nem tudunk megadni [3].

Minőségi szempontok

Adott vállalatnál a létező minőségbiztosítási módszerek és a figyelembe vett szabványok is nyújthatnak támpontot a várható minőségre. Illetőleg a fejlesztő cég CMM[9] besorolása is.

Hiba mérték

Ez lenne a legfontosabb besorolás, mert ez ad pontos megbízhatósági mutatót. A megbízhatóság időfüggő. Megbízhatósági szempontból a következő tesztelési típusok jöhetnek szóba:

1. **Stressz tűrési teszt.** A szoftvert (rendszert) olyan körülmények közé helyezi amely annak működését jelentősen befolyásolja, például az erőforrásokat csökkenti.

[6] LOC – Line Of Code – A program kódsorainak száma
[7] KLOC – Kilo Line Of Code – A program kódsorainak száma ezer sorokban kifejezve
[8] IPC – Inter Process Communication, Folyamatok közötti kommunikáció
[9] CMM – Capability Maturity Model, Képesség-érettség modell

2. **Terhelési és stabilitási teszt.** A szoftvert maximális terheléssel hosszú ideig vizsgálja.
3. **Megbízhatósági teszt.** Hasonló az előzőhöz, de átlagos terhelés mellett végzik a tesztelést.
4. **Regressziós teszt.** Azt vizsgálja, hogy a szoftver stabil marad-e, ha új elemek jelennek meg a rendszerben, illetve a rendszer valamely része hibásan kezd működni.

Ezeknek a teszteknek az a legnagyobb baja, hogy hosszú ideig tartanak, hiszen statisztikai jellegű eljárások. A kapott eredményeket a következő mérőszámokban fejezik ki [9]:

1. A hiba bekövetkezési valószínűsége adott igényre (POFOD[10]). Alapvetően egy adott bemenetre mekkora a hiba bekövetkezési valószínűsége.
2. A hiba bekövetkezési frekvenciája (ROCOF[11]). Ezt adott időtartamra adják meg.
3. Az átlagos hiba bekövetkezés közti eltelt idő (MTBF[12], vagy MTTF[13]).
4. Rendelkezésre állás. A rendszer egy adott időtartam alatt mennyi ideig áll rendelkezésre és mennyi ideig tart a karbantartása.

További probléma a tesztelési környezet helyessége. Nyilvánvaló, hogy a hosszú távú tesztelést teljes egészében nem végezheti ember. Ezt automatizálni kell. Ha a teszt környezet hiányos, akkor bizonyos szoftver részletek nem kerülnek tesztelésre, illetőleg az automata teszter nem vesz észre eltéréseket.

Hiba és eltérés

Sajnos a magyar nyelvben a failure és a fault angol kifejezésekre a fordítás minden esetben a hiba. Ezért kissé önkényesen fordítottuk a failure kifejezést eltérésnek és a fault kifejezést hibának.

Már a definícióból kiderül, hogy egy működési eltérés nem minden esetben okoz hibát. Elképzelhető, hogy egy avatott szemlélő észreveszi, hogy az adott rendszer kis mértékben eltér az előírt működéstől, de ez nem biztos, hogy a rendszer eredő működési jellemzőiben látszik. Az az eltérés, amely már rendszer szinten látható, az már hiba.

A hibák besorolása a következő lehet: [2]

- nem jelentős (jelentéktelen),
- jelentős (kritikus),
- ismert,
- ismeretlen.

Nem jelentős hiba esetén, akár ismert akár ismeretlen nem várhatunk komoly rendszer működési zavart. Ha azonban a hibát sikerült felderíteni, akkor záros határidővel azt javítani kell. Nem biztonságkritikus esetben egy kockázatelemzés után előfordul, hogy a hiba javítatlan marad. Példaként említhetünk kisebb színezési hibákat.

[10] POFOD – Probability of Failure on Demand, A hiba bekövetkezési valószínűsége adott igényre
[11] ROCOF – Rate Of Failure Occurence Frequency, Hiba bekövetkezési frekvenciája
[12] MTBF – Mean Time Between Failures, A meghibásodások között eltelt idő
[13] MTTF – Tean To Failure, Egy meghibásodásig eltelt átlagos idő

Nem jelentős hiba esetén, akár ismert akár ismeretlen nem várhatunk komoly rendszer működési zavart. Ha azonban a hibát sikerült felderíteni, akkor záros határidővel azt javítani kell. Nem biztonságkritikus esetben egy kockázatelemzés után előfordul, hogy a hiba javítatlan marad. Példaként említhetünk kisebb színezési hibákat.

Valahol a hiba és az eltérés közé sorolnánk a Boeing 787 Dreamliner generátor problémáját. Egyértelműen komoly működési probléma, de ehhez a hajtóműnek valószínűtlenül hosszú ideig kell működnie. Ezen hiba javítás meg kell oldani, de nem szükséges a típus időszakos üzemeltetésének felfüggesztése[7].

Kritikus hiba esetén a javítás nem halasztható, azt azonnal végre kell hajtani. A legnagyobb biztonsági problémát az ismeretlen kritikus hibák okozzák. Nem ismert, hogy milyen körülmények között következnek be és nem ismert hatásuk sem. Lásd Sevilla-i katasztrófa.

Első pillanatra nyilvánvalónak tűnik a hibák azonnali javítása, azonban a javítás is hordozza a hiba lehetőségét. Rendkívül fontos a javított szoftver alapos validálása [2].

Természetes módon felmerülő kérdés, hogy miért nehéz az eltérések és a hibák felderítése.

A válasz a következő pontokban adható meg:

1. a hiba csak bizonyos esetekben számos körülmény együttes fennállása esetén következik be;
2. a hiba normális körülmények között nem következik be, vagy hatása észrevétlen marad;
3. a hiba folyamatosan fejlődik, majd egy adott szint elérésekor rendszerszintű problémát okoz.

Az első eset tipikus konkurens rendszerek esetén. Hibás tervezés esetén adott konstellációban a teljes rendszer, vagy egyes részei blokkolttá válnak.

A második esetben a rendszer a lehetséges állapottér olyan állapotába kerül, amelyre nincs felkészítve és rosszabb esetben erre választ is ad. Ilyen szerencsétlen eset lehet az úgynevezett halott kód futása.

A harmadik hiba nagyon banális lehet. A rendszer valamilyen erőforrást folyamatosan és észrevétlenül fogyaszt. Erre tipikus példa a naplózás háttértárra, amely a tárolóterület elfogyásához vezet, vagy a memóriaszivárgás.

A MEGBÍZHATÓSÁG NÖVELÉSE

Elsődleges cél a megbízhatóság növelése. A lehetséges módszerek a következők:

1. Helyes tervezési módszerek alkalmazása. Biztonságtechnikai és kódolási eljárások betartása vétele, mint például MISRA, illetve az ide vonatkozó szabványok figyelembe vétele IEC 61508 [2].
2. Az előállított adatok és programok minél kimerítőbb tesztelése akár azon az áron is, hogy egy időben több párhuzamosan futó tesztelési projekt fut. Fokozottan érvényes ez a hosszú távú, terheléses és regressziós tesztekre.

3. Az előállított kód legyen hibatűrő, kisebb eltéréseket és hibákat képes legyen korrigálni azért, hogy ezek ne vezessenek komolyabb rendszer működés eltéréshez [9].
4. Olyan szoftver egységek használata, amelyet már többször felhasználtak és hibátlannak minősültek. Ez sem tekinthető végső megoldásnak. Ennek oka, hogy a felhasznált elemek olyan körülmények közé kerülhetnek, amelyben nem működhetnek helyesen.
5. Az előállított kód tartalmazzon biztosítékokat az esetleges komolyabb hibák időben történő felderítésére és ezek hibák kialakulásának megakadályozására.

A probléma továbbra is fennáll, minden redundancia jellegénél fogva növeli a komplexitást. A komplexitás növekedése viszont a hiba lehetőségét hordozza magában.

Volt már arra példa, hogy egy jól működő rendszert a biztonságot szolgáló szoftver elem tett használhatatlanná.

Következtetések és összefoglalás

A szoftver megbízhatóság valószínűségi jellemző. Nehéz olyan valószínűségi modellt találni, amely jól becsülhetővé teszi az értékét. Több szempontból vizsgálva kijelenthetjük, hogy egységes és minden szempontból megfelelő elméletet nem tudunk megnevezni.

Különböző módszerekkel, szabályokkal és teszteléssel a megbízhatóság javítható, de a tapasztalat azt mutatja, hogy hasonlóan a más ember által előállított rendszerekhez a szoftverek megbízhatósága nem lehet 100%-os.

FELHASZNÁLT IRODALOM

[1] VAN SOLINGEN, R., BERGHOUT, E. The Goal/Question/Metric Method: A Practi-cal Guide for Quality Improvement and Software Development. McGraw-Hill Inter-national. (1999), pp. 56-67

[2] TOKODY DÁNIEL, SCHUSTER GYÖRGY, PAPP JÓZSEF: Study of How to Implement an Intelligent Railway System in Hungary In: Anikó Szakál SISY 2015 : IEEE 13th nternational Symposium on Intelligent Systems and Informatics: Proceedings. Subotica, Szerbia, 2015.09.17-2015.09.19. Subotica: IEEE Hungary Section, 2015. pp. 199-204.

[3] LONG, J (ed.) Metrics Data Program, National Aeronautics and Space Administration (2008), (online), url: http://mdp.ivv.nasa.gov/index.html (2015.12.28)

[4] NORMAN F. SCHNEIDEWIND: Reliability Modeling for Safety Critical Software, IEEE Transactions on Reliability, Vol. 46, No.1, 1997.03, pp.88-98

[5] JAD MOUAWAD: F.A.A. Orders Fix for Possible Power Loss in Boeing 787 (online), url: http://www.nytimes.com/2015/05/01/business/faa-orders-fix-for-possible-power-loss-in-boeing-787.html (2015.04.30)

[6] JENS FLOTTAU, TONY OSBORNE: Software Cut Off Fuel Supply In Stricken A400M (online),url: http://aviationweek.com/defense/software-cut-fuel-supply-stricken-a400m (2015.05.23)

[7] AVIATION SAFETY NETWORK: Accident description,(online), url: http://aviation-safety.net/database/record.php?id=20150509-0 (2016.01.17)

[8] J. L. LIONS: ARIANE 5 Flight 501 Failure Report by the Inquiry Board (online), url: https://www.ima.umn.edu/~arnold/disasters/ariane5rep.html (2016.02.20)

[9] JIANTAO PAN: Software Reliability (online), url: https://users.ece.cmu.edu/~koopman/des_s99/sw_reliability/ (2016.02.21)

A SOFTWARE RELIABILITY

Software reliability is the error-free working probability of software for a specified period of time under well-defined environment. Software is critical success factor consequently it is inevitable. Considering this software reliability has a strong effect on the reliability of the whole system. Another tendency is that the complexity of software is increasing in order that software could eliminate errors of the given system, environment and users. That is well-known fact that more complex systems have more possibility of errors. Testing of reliability has well-defined methods in case traditional systems. For example fatigue and wearing of a mechanical part or features of a lubricant can be calculated well. These methods are not suitable for estimation of software reliability. In case of traditional systems the conditions of operation and maintenance have influence on reliability of the system in case of software systems the correctness of the design has the main emphasis.

Keywords: *Software, reliability, system, safety critical, testing*

Dr. SCHUSTER György PhD Intézetigazgató, egyetemi docens Óbudai Egyetem Kandó Kálmán villamosmérnöki Kar Műszertechnikai és Automatizálási Intézet schuster.gyorgy@kvk.uni-obuda.hu orcid.org/0000-0002-8573-3670	Dr. SCHUSTER György PhD Associate professor Director of Institution of Instrumentation and Automation Kandó Kálmán Facutry of Elecrical Engineering Universitas Budensis schuster.gyorgy@kvk.uni-obuda.hu orcid.org/0000-0002-8573-3670
TERPECZ Gábor (MSc) Mérnöktanár Óbudai Egyetem Kandó Kálmán villamosmérnöki Kar Műszertechnikai és Automatizálási Intézet terpecz.gabor@kvk.uni-obuda.hu orcid.org/0000-0001-7899-2837	TERPECZ Gábor MSc. EE. Engineer Teacher of Institution of Instrumentation and Automation Kandó Kálmán Facutry of Elecrical Engineering Universitas Budensis terpecz.gabor@kvk.uni-obuda.hu orcid.org/0000-0001-7899-2837
TOKODI Dániel (MSc) okl. villamosmérnök MÁV Szolgáltató Központ Zrt. tokodi.daniel@bgok.hu orcid.org/0000-0002-9984-0434	TOKODI Dániel MSc. EE. MÁV Service Centre tokodi.daniel@bgok.hu orcid.org/0000-0002-9984-0434

http://www.repulestudomany.hu/folyoirat/2016_2/2016-2-13-0316_Terpecz_Gabor_et_al.pdf