Szoftver megbízhatóság

KESSELYÁK PÉTER
BHG Fejlesztési Intézet

ÖSSZEFOGLALÁS

A bemutatott szoftver megbízhatósági modell célja az, hogy a mikroprocesszor-vezérelt rendszerek megbízhatóságának elemzéséhez hasznos, szemlélet formáló segédeszközt bitosítson. A modell a szoftver programcsomag működési környezetét sokdimenziós állapottérként kezeli, amelyben a szoftver folyamatok a bennük rejlő hibaforrásokkal együtt "átlátszóvá", könnyen érthetővé válnak. Az úgynevezett tesztelési horizonton belül az állapottér magja szoftver hibaforrásoktól mentes. Az állapottér következő, külsőbb övezete, amely a tesztelési horizont és a fejlesztési horizont között helyezkedik el, viszonylag kevés szoftver hibaforrással terhelt, míg a működési állapottér legkülsőbb, perifériális övezetében - a fejlesztési horizonton kívül - a "fekete doboz" állapota a jellemző, tele ismeretlen szoftver hibaforrással.

1. Bevezetés

Napjainkban a szoftver milyensége egyre jelentősebb szerepet játszik a rendszer-megbízhatóság megítélésében. Az elmúlt 20 év során számos szoftver-megbízhatósági modellt fejlesztettek ki világszerte, amelyeknek mindegyike - csaknem kivétel nélkül - egyetlen gyakorlati kérdés megoldását tűzte ki célul. Nevezetesen azt, hogy hogyan lehet a hibákat valamely programból lépésről lépésre kiküszöbölni, és ugyanakkor a programcsomagban bennmaradó hibák számát statisztikai megfontolások alapján becsülni. A teljesség igénye nélkül utalunk itt Remus, Jelinski-Moranda, Musa, Nelson és Littlewood-Verall modelljeire. [2], [3], [4]. A továbbiakban nem ezekkel a modellekkel kívánunk foglalkozni, hanem egy olyan általános g o n d o l k o d á s m ó d o t szeretnénk bemutatni, amely a szoftver-megbízhatóság lényegének megértéséhez és problémáinak kezeléséhez hasznos segítséget nyújt, azokat könnyen áttekinthetővé teszi. Ez a gondolkodásmód - vagy másképp: szemlélet modell - Hermann Kopetz szoftver-filozófiáján alapszik [1], amelynek azonban számos tekintetben továbbfejlesztését jelenti.

2. Hardver és szoftver jellemzők szembeállítása

Mindenek előtt szeretnénk rávilágítani, miben áll az alapvető különbség a hardver és a szoftver megbízhatósága között. Ehhez célszerű szembeállítani egymással néhány jellemzőjüket:

- A hardver (továbbbiakban: HW) mindig alá van vetve fizikai öregedési folyamatoknak és a hibák az i d ő f o l y a m á n keletkeznek: a szoftver (a továbbiakban: SW) ezzel szemben mindig "fiatal" marad és a hibagenerátorok [az úgynevezett "szoftver bag"-ok] kezdettől fogva jelen vannak a programcsomagban;
- A HW esetében abszolút hibamentes működés elméletileg sem lehetséges, a hibamentes működés valószínűsége - bármilyen rövid időtartamra vonatkoztatva tekintjük is - mindig k i s e b b, mint 1; a SW esetében viszont a programfutás - bizonyos feltételek mellett - lehet abszolút hibamentes;
- HW esetében a hiba mindig egyértelműen hozzárendelhető valamely alkatrészhez vagy alkatrészcsoporthoz, míg a SW hibák nem mindig rendelhetők hozzá egyértelműen valamely programlépéshez, programszegmenshez vagy programcsomaghoz;
- A HW egységek soros/párhuzamos konfigurációjának tömbvázlata alapot nyújt az egységekből felépülő rendszer megbízhatóságának elemzéséhez; a SW programcsomagokból felépülő rendszer megbízhatósága ezzel szemben nem osztható ki az egyes program részek között, hiszen az egyes programszegmensek vagy programcsomagok soros/párhuzamos konfigurálása értelmetlen dolog; ezért a HW megbízhatóság-elemzésben jól bevált Boole-algebra szabályai nem alkalmazhatók;

A fentiekben bemutatott HW-SW szembeállítást az *1. táblázat* szemléletesen foglalja össze.

Annak ellenére, hogy a HW és SW megbízhatóság alapvetően különböző megközelítést igényel, a kettő nem független egymástól. Mindig létezik egy olyan működő r e n d s z e r, amely a hardvert, a softvert - sőt, a működtető személyzetet is - magában foglalja.

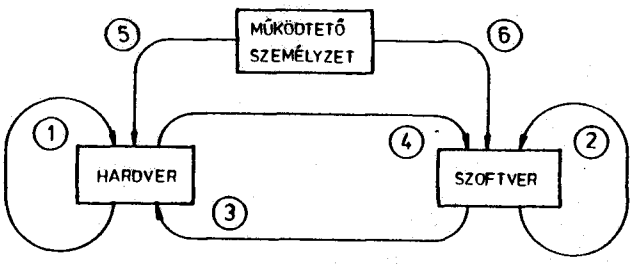
3. Kölcsönhatás a hardver, szoftver és az üzemeltető személy között

A HW, SW és a működtető személyzet közötti kölcsönhatásokat a *1. ábrán* nyilakkal ábrázoltuk.

Beérkezett: 1990. II. 20. (#)

KESSELYÁK PÉTER

Matematika-fizika szakon szerzett diplomát 1958-ban. 1959 óta a BHG Híradástechnikai Vállalat fejlesztő mérnöke. Több éven át dolgozott műszaki-tudományos együttműködés keretében Dél-Kínában, majd Kubában, híradástechnikai gyártmányok trópusállósági és megbízhatósági vizsgálatait végezve. Fő munkaterülete a rendszer-megbízhatóság tervezése. Tagja az Európai Minőségügyi Szervezet és az IEC 56. Megbízhatósági Szakbizottsága hazai munkacsoportjának. 1984-ben megkapta az Európai Minőségügyi Szervezet (EOQC) nívódíját. 1986-ban a HTE Puskás Tivadar díjjal tüntette ki. Az UNIDO felvette szakértői nyilvántartásba.



1. ábra Működő rendszerek egyesített hardver-szoftver megbízhatósági modellje
KÖLCSÖNHATÁSOK A RENDSZER RÉSZEI KÖZÖTT: (1) "Tiszta" hardver megbízhatóság; (2) "Tiszta" szoftver megbízhatóság; (3) Szoftver befolyása a hardver megbízhatóságára (Pl. érintkező kopás gyakori hibás vezérlés miatt); (4) Hardver befolyása a szoftver megbízhatóságára (Pl: zavarimpulzusok okozta adatvesztés); (5) Emberi tényezők befolyása a hardver megbízhatóságára; (6) Emberi tényezők befolyása a szoftver megbízhatóságra.

INTEGRÁLT HARDVER MEGBÍZHATÓSÁG összetevői: (1) + (3) + (5)

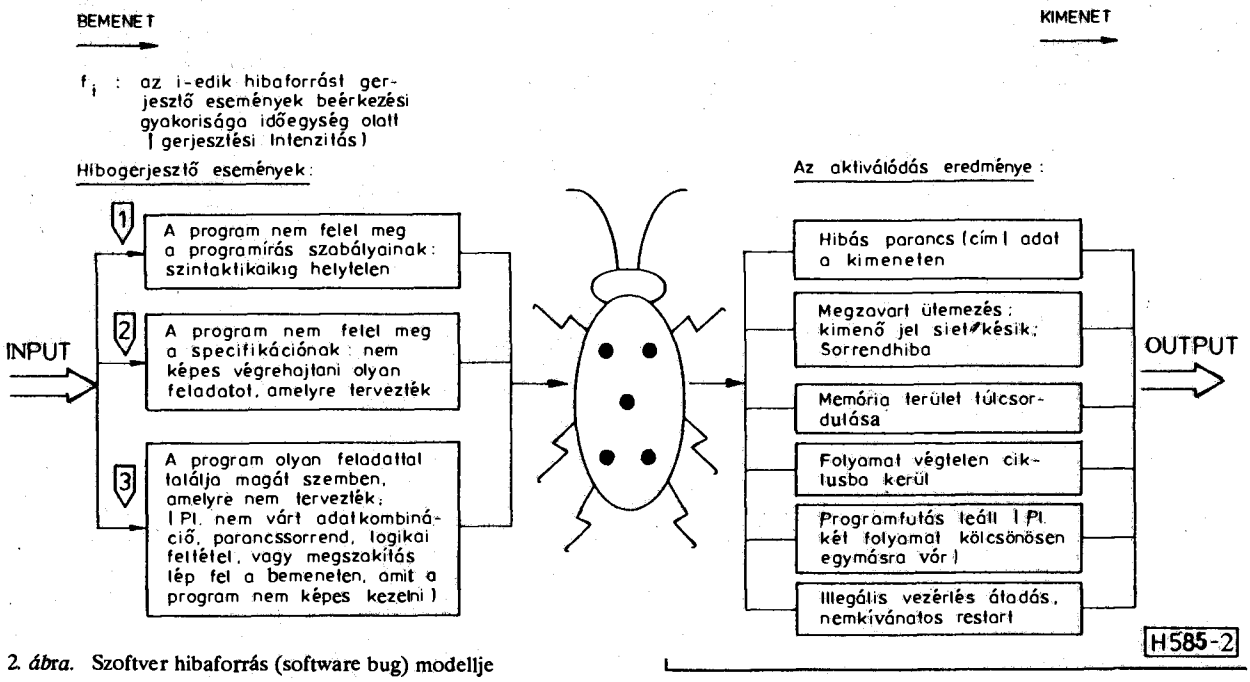
INTEGRÁLT SZOFTVER MEGBÍZHATÓSÁG összetevői: (2) + (4) + (6)

Az ábrán [1]-el jelölt visszacsatoló hurok a t i s z t a hardver megbízhatóságot jelképezi. Hasonlóképpen a [2]-es visszacsatoló hurok á t i s z t a szoftver megbízhatóságot szimbolizálja. A [3]-as nyíl a szoftvernek a hardverre gyakorolt hatását fejezi ki (pl. érintkező kopás gyakori hibás SW vezérlés hatására bekövetkező kapcsolásnál). A [4]-es nyíl a hardvernek szoftverre gyakorolt hatását testesíti meg (pl. elektromágneses zavarjelek okozta adatvesztés a memóriában, amely a programfutást tévútra vezeti) Az [5]-ös és [6]-os nyíl a személyzetnek a HW-re illetve SW-re gyakorolt (káros vagy hasznos) befolyását reprezentálja.

A vázolt kölcsönhatások figyelembe vételével juthatunk el az i n t e g r á l t s z o f t v e r - m e g b í z h a t ó s á g fogalmához, amely a 2-es + 4-es + 6-os összetevőket foglalja magába. Hasonlóképen, az i n t e g r á l t h a r d v e r - m e g b í z h a t ó s á g az 1-es + 3-as + 5-ös összetevőkből áll össze.

1. táblázat
A hardver- és szoftver-megbízhatóság jellemzőinek összehasonlítása

HARDVER	SZOFTVER
Mindig alá van vetve fizikai öregedésfolyamatoknak	Nem öregszik
Meghibásodás az idő folyamán keletkezik	A szoftver hibaforrás (BUG) kezdettől fogva állandóan jelen van, amíg ki nem küszöbölik
Abszolút hibamentes működés elvileg lehetetlen	A programfutás bizonyos körülmények között lehet abszolút hibamentes
A hiba egyértelműen hozzárendelhető valamely alkatrészhez vagy szerelvényhez	A szoftver hibaforrás (BUG) általában nem rendelhető hozzá egyértelműen egy programszegmenshez vagy csomaghoz
Az egységek kapcsolási (soros/párhuzamos) tömbvázlata megkönnyíti a rendszer-megbízhatóság elemzését; a rendszer-megbízhatóság kiosztható a funkcionális egységek között	A rendszer-megbízhatóság nem osztható ki az egyes programszegmensek között; soros-párhuzamos kapcsolati viszony értelmetlen; a Boole-algebra nem alkalmazható a rendszer-megbízhatóság elemzésére



2. ábra. Szoftver hibaforrás (software bug) modellje

Ez a fajta integráció gyakorlati szempontból igen célszerű, mivel a 2-es, 4-es és 6-os típusú hatásmechanizmusok mindig szoftver szimptómájú hibát okoznak, amelynek elhárítása is elsődlegesen SW eszközökkel történik, nevezetesen adatcserével, program módosítással és/vagy restarttal. Megjegyezzük, hogy a 4-es hatásmechanizmus esetében előfordulhat, hogy a hibaelhárítás - SW eszközökön túlmenően – egyes HW jellemzők beállítását is igényli. Rendszerint azonban csak igen alapos hibaelemzéssel lehet pl. a HW zavarimpulzusok következményeként előálló programfutási hibákat a valódi, tiszta szoftver hibáktól megkülönböztetni.

Hasonlóan lehet érvelni az integrált hardver-megbízhatóság fogalmának gyakorlati hasznossága mellett is, mivel az 1-es, 3-ás és 5-ös hatásmechanizmusokban közös az, hogy mindig hardver hibát okoznak, amely a HW javítása révén hárítható el.

4. A szoftver hibaforrás (SOFTWARE BUG) fogalma

A szoftver hibaforrás (SW BUG) egy állandó, a programhoz és annak működési környezetéhez hozzárendelt zavarforrás, amely a program ismételt futtatásával ismételten aktivizálható. Megjegyezzük, hogy az angol BUG (ejtsd: bag) szó eredetileg bogarat jelent. Minden egyes szoftver hibaforrás (bug) megfelel egy olyan parancs vagy programszegmens hiányának, amelynek a lefuttatása - bizonyos körülmények között - a szoftver hibás működését képes (lenne) megakadályozni.

A szoftver hibás működése lehet pl.: valamely folyamat leállása, végtelen ciklusba esés, nemkívánatos restart, a programfutás eltévelyedése, vagy hibás eredmény szolgáltatása.

2. táblázat

A szoftver hibaforrás (software BUG) fogalma

Definíció: A szoftver hibaforrás (software bug) egy állandó, a programhoz és annak működési környezetéhez tartozó zavarforrás, amely a program ismételt futtatásával ismételten aktivizálható.
Ekvivalencia axióma: Minden egyes szoftver hibaforrás (software bug) megfelel egy olyan parancs vagy programszegmens hiányának, amelynek a futtatása - bizonyos körülmények között - a szoftver hibás működését képes (lenne) megakadályozni.

Lényegében minden szoftver hibaforrás (bug) valaminek a hiányát jelenti, ezért általában nehéz dolog konkrétan meghatározni, hogy valójában hol található. Fizikai hasonlattal élve: kettős természetű, mint a fény, amely fizikai kölcsönhatásokban anyagi részecskéhez hasonlóan, fotonként jelenik meg a kölcsönhatás helyszínén, egyébként pedig hullámtermészetű és pontosan meghatározható helye nincs.

Azt az állítást, hogy a szoftver hibaforrás mindig egyenértékű bizonyos programutasítás vagy utasítások hiányával, direkt módon sem bizonyítani, sem tagadni nem lehet. Ezt az állítást a szoftver hibaforrás ekvivalencia axiómájának tekinthetjük.

A továbbiakban vizsgáljuk meg részletesebben a szoftver hibaforrás modelljét, és tekintsük a 2. ábrát.

A bemeneti oldalon található az (i-edik) szoftver hibaforrást aktiváló (gerjesztő) események f_i intenzitása (átlagos ismétlődési frekvenciája). A gerjesztés okai három csoportba sorolhatók. A legegyszerűbb eset az, amikor a program szintaktikailag helytelen, vagyis nem felel meg a programírás szabályainak. A második eset akkor áll elő, ha a program - jóllehet szintaktikailag helyes - nem felel meg a specifikációnak, vagyis előírt feladatot előírt programfutási feltételek mellett nem képes megoldani. Végül a harmadik eset akkor következik be, ha vagy a megoldandó feladat, vagy a programfutás környezeti feltételei kívül esnek a specifikáción, vagyis a program olyan feladattal kerül szembe, amelynek megoldására nem is tervezték. (Ez természetesen eredhet a specifikáció hiányosságából is). Megoldhatatlan feladatot jelenthet a program számára pl., ha a bemeneten nem várt adatkombináció, nem várt parancs sorrend, nem várt logikai kombináció vagy megszakítás igény jelentkezik, amit a program nem képes kezelni.

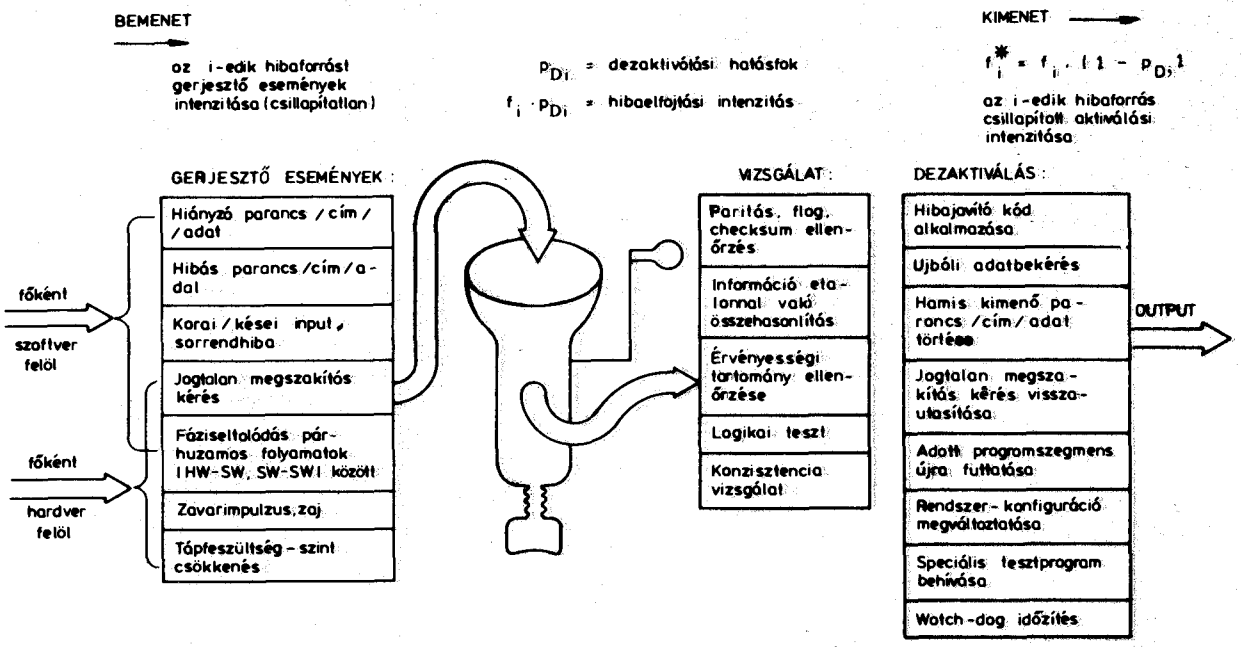
A szoftver hibaforrás kimenetén a gerjesztés "eredménye" található. A leggyakoribb kimeneti események: hamis információ, rossz időzítés (a kimenő jej túl korán vagy túl későn jelenik meg), hibás parancs sorrend, végtelen ciklusba vezérlés, vagy a programfutás leállása (pl. két folyamat egymásra várása miatt), illegális vezérlés átadás, nem kívánt restart.

Ezzel rövid jellemzést adtunk egy - a szoftver megbízhatóság szempontjából kulcsfontosságú - fogalomról, a szoftver hibaforrásról vagy "SOFTWARE BUG"-ról. Nem kevésbé fontos azonban egy másik nélkülözhetetlen fogalom - a softver hibatűrő képesség, a "SOFTWARE ROBUSTNESS" - értelmezése sem.

5. A szoftver hibatűrő képesség fogalma

A szoftver hibatűrő képesség vagy szoftver "robusztusság" (software robustness) a szoftverba beépített olyan képesség, amelynek célja, hogy a szoftver hibaforrások ("bug"-ok) aktiválódását megelőzze vagy elfojtsa. Elméletileg minden egyes szoftver hibaforráshoz hozzárendelhető egy hibagátló forrás úgy, hogy a kettő együtt pár t képezzen. Az *3. sz. ábra* a hibatűrő képességet magában hordozó hibagátló forrás modelljét mutatja.

A hibagátló forrás bemeneti oldalán található az (i-edik) szoftver hibaforrást aktiváló (gerjesztő) események f_i intenzitása (átlagos ismétlődési frekvenciája).



3. ábra. A szoftver hibatűrő képessége: hibagátló forrás (Robustness Source) modellje

H586-3

Az aktiváló események többnyire az alábbi választékból kerülnek ki:

- hiányzó cím/adat/utasítás a bemeneten;
- hibás cím/adat/utasítás a bemeneten;
- korai/kései bemenőjel, hibás sorrend;
- fáziseltolódás hardver/szoftver vagy szoftver/szoftver párhuzamos folyamatok között;
- jogtalan megszakítás kérés;
- zavarjel, zaj;
- tápfeszültség csökkenés.

A felsorolt események többsége szoftver folyamatokból vagy átmeneti hardver hibajelenségekből származhat.

A hibagátló forrás a vele "párhuzamosan kapcsolt" hibaforrás bemenetén jelentkező hibagerjesztő eseményeket tesztelés útján "igyekszik" kiszűrni. Ilyen teszteljárások pl.:

- a paritás/flag/checksum ellenőrzés;
- névleges adattal való összehasonlítás;
- érvényességi tartomány vizsgálat;
- logikai/konzisztencia vizsgálat.

A vizsgálat eredményétől függően a hibagátló forrás "intézkedéseket foganatosíthat" a hibaforrás gerjesztésének megszüntetésére - más szóval - dezaktiválására. A dezaktiválás pl. az alábbi módon történhet:

- hibajavító kód alkalmazása;
- újbóli adatkérés;
- hibás kimeneti adat törlése;
- jogtalan megszakítás kérés visszautasítása;
- adott programszakasz vagy csomag újra futtatása;
- rendszer-konfiguráció megváltoztatása;
- speciális tesztprogram behívása;
- watch-dog időzítés;
- alarmjelzés az operátor számára.

Tudomásul kell venni, hogy a dezaktiválási eljárás nem minden esetben lesz sikeres. Előfordulhat, hogy a hibaforrás gerjesztését a hibagátló forrás nem képes elfojtani. A dezaktiválás sikerességét egy hatásfok jellegű mennyiséggel mérhetjük, a P_{Di} dezaktiválási vagy hiba elfojtási aránnyal, amely az i-edik szoftver hibaforrás sikeresen elfojtott gerjesztéseinek aránya az összes gerjesztéshez viszonyítva (elméletileg a sikeres elfojtás valószínűsége).

Az i-edik szoftver hibaforrás (BUG) és vele "párhuzamosan" kapcsolt szoftver hibagátló forrás (ROBUSTNESS SOURCE) közös kimenetén a gerjesztő események csillapított f_i^* aktiválási intenzitása jelennek meg:

$$f_i^* = f_i \cdot (1 - p_{Di}) \qquad (i = 1,...,N) \qquad (1)$$

ahol f_i az i-edik hibaforrás bemeneti oldalán jelentkező gerjesztő események csillapítatlan intenzitása (átlagos ismétlődési frekvenciája), az $1 - p_{Di}$ tényező pedig az i-edik szoftverhibaforrás aktiválódási hatásfoka, amit a továbbiakban Q_i-vel jelölünk.

$$Q_i = 1 - p_{Di} \qquad (i = 1,...,N) \qquad (2)$$

6. A hibaforrás és a hozzá tartozó hibagátló forrás együttélési modellje

A szoftver hibaforrás (SOFTWARE BUG) és a hozzá tartozó hibagátló forrás (ROBUSTNESS SOURCE) egymástól elválaszthatatlan kapcsolatban vannak, ahogyan azt a 4. ábra mutatja.

A modell bemenetén a hibagerjesztő események f_i csillapítatlan intenzitása jelenik meg, amely egyformán eljut a hibaforráshoz és a hibagátló forráshoz. A hibagátló forrás a gerjesztő esemény érzékelésekor vezérlő jelet ad a hibaforrással "sorba" kapcsolt csillapító tag-

nak. A csillapító tag kapcsolóként működik, amely a hibaforrás gerjesztése nyomán keletkező szoftver hibát vagy átengedi, vagy blokkolja (elfojtja). A csillapító tag kimenetén az i-edik szoftver hibaforrás csillapított $f_i^* = f_i \cdot Q_i$ aktiválási intenzitása fog megjelenni.

7. A szoftver megbízhatóság általános jellemzői

A szoftver megbízhatósági modell lépésről lépésre történő felépítése során eddig szándékosan elkerültük magának a szoftver hibának a meghatározását, mivel ennek megértéséhez több más fogalom előzetes bevezetése volt szükséges. Ezek birtokában most már definiálhatók a szoftver megbízhatóság alábbi jellemzői:

- Szoftver hibának nevezzük a szoftver hibaforrás (BUG) egyszeri aktiválását.
- Az i-edik típusú szoftver hibák száma egyenlő az i-edik szoftver hibaforrás (BUG) ismételt aktiválásainak számával.
- Az i-edik fajta szoftverhibaintenzitás a egyenlő az i-edik szoftver hibaforrás csillapított aktiválási intenzitásával. Képletben:

$$Z_{swi} = f_i^* = f_i \cdot Q_i \qquad (3)$$

vagyis a szoftver hibaintenzitás egyenlő a hibaforrás gerjeszési intenzitásának és az aktiválódási hatásfoknak a szorzatával.

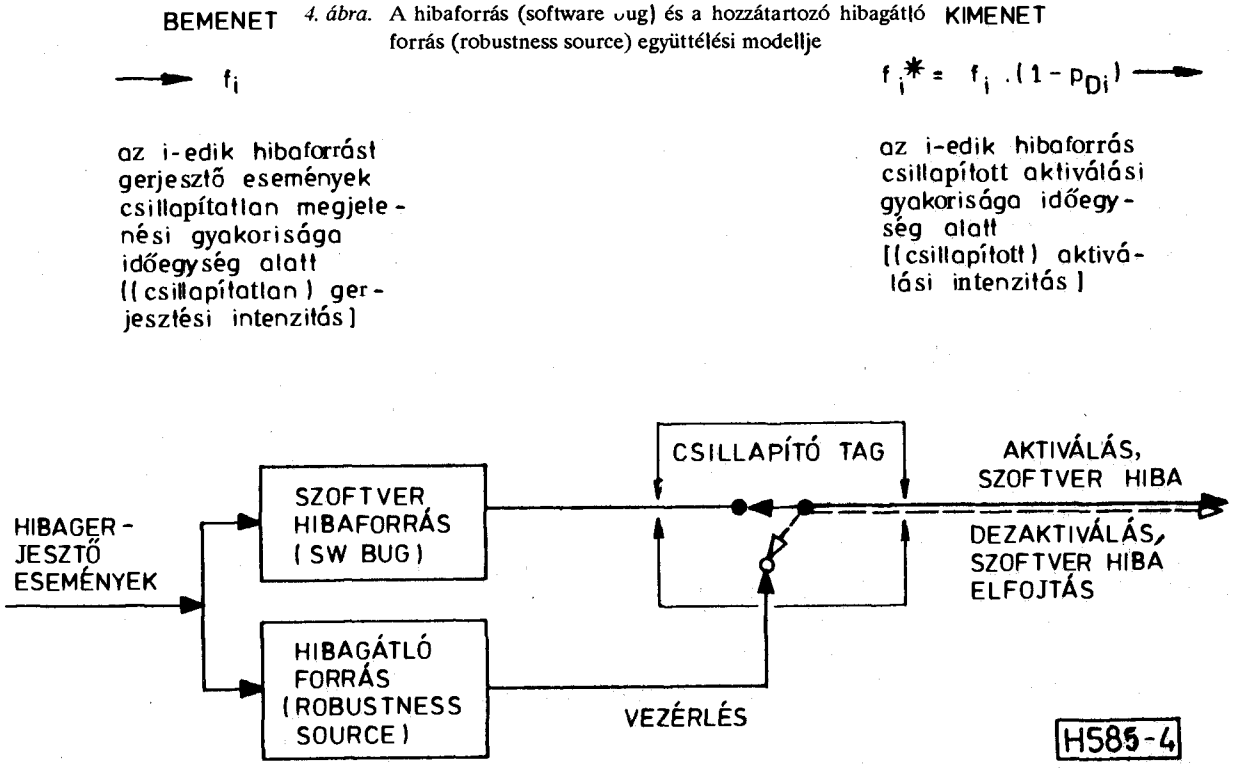
Jelöljük valamely programcsomagban jelenlévő szoftver hibaforrások számát N-el, ezen belül az egyes hibaforrások sorszámát i-vel. (i = 1,...,N). Jelöljük továbbá a programcsomag egy konkrét alkalmazását A-val.

Minden program-modulnak, szegmensnek alkalmazásonként más és más lehet a futási gyakorisága. Ezzel összefüggésben nyilvánvaló az is, hogy az egyes modulokhoz, szegmensekhez hozzárendelhető hibaforrások (BUG-ok) aktiválási gyakorisága is alkalmazásonként különböző lehet. A programcsomag minden egyes A alkalmazásához megadhatók a programcsomagban lévő szoftver hibaforrások $f_1(A), f_2(A),...,f_i(A),...,f_N(A)$ gerjesztési intenzitásai, amelyek együtt egy N dimenziós $\vec{f}(A)$ vektorban foglalhatók össze. Ezt az $\vec{f}(A)$ vektort alkalmazásfüggő hibagerjesztési vektornak nevezzük. Ez a vektor független a programcsomagba illetve a rendszerbe beépített szoftver hibatűrő képességtől (a robustness-től) és kizárólag az adott A alkalmazástól függ. Másfelől jelöljük á programcsomagban lévő szoftver hibaforrások aktiválódási hatásfokát $Q_1,Q_2,...,Q_i,...,Q_N$-nel. A programcsomag szoftver hibatűrő képessége (robusztussága) a

$$\vec{Q}(R) = (Q_1,Q_2,...,Q_i,...,Q_N) \qquad (4)$$

hibafojtási vektor segítségével jellemezhető. Ez a hibafojtási vektor független a programcsomag konkrét alkalmazásaitól és csupán a programcsomagba, operációs rendszerbe és a hardver tesztelő eszközökbe beépített hibatűrő képességtől, vagyis a szoftver "robusztusságától" függ. (A $\vec{Q}$ (R) jelölésben erre utal az R argumentum).

Jelöljük a programcsomag közepes futásidejét az A alkalmazásban T(A)-val. Akkor az A alkalmazásban egyetlen programlefutás alatt a keletkező szoftver hibák várható számát az alábbi összefüggés határozza meg:



4. ábra. A hibaforrás (software bug) és a hozzátartozó hibagátló forrás (robustness source) együttélési modellje

$$r(A) = (\vec{f}(A), \vec{Q}(R)) \cdot T(A) =$$

$$= T(A) \cdot \left[\sum_{i=1}^{N} f_i(A) \cdot Q_i \right] \qquad (5)$$

Ez azt jelenti, hogy a szoftver hibák várható száma programfutásonként két vektor skaláris szorzatával arányos: az egyik az A alkalmazástól függő hibagerjesztési vektor, a másik a rendszer hibatűrő képességétől függő hibafojtási vektor, az arányossági tényező pedig a programfutás T(A) várható időtartama.

Egy további, nagyon hasznos szoftver megbízhatósági jellemzőt szolgáltat az /5/ összefüggés reciproka:

$$n(A) = 1/r(A) \qquad (6)$$

n(A) nem más, mint a szoftver hibák közti programfutások átlagos száma az A alkalmazásban. A gyakorlatban ennek a megbízhatósági jellemzőnek a jelentősége a legnagyobb. A most bevezetett megbízhatósági jellemzőkről a 3. táblázat nyújt áttekintést.

3. táblázat

Szoftver megbízhatósággal kapcsolatos mennyiségi jellemzők áttekintése

N =	Adott programcsomaghoz és annak szoftver környezetéhez tartozó szoftver hibaforrások (SW bug-ok) száma;
i =	Az egyes szoftver hibaforrások (SW bug-ok) azonosító sorszáma (i = 1,...,N)
A =	Az adott programcsomagnak egy konkrét alkalmazása;
f_i =	Az i-edik szoftver hibaforrás gerjesztési intenzitása;
p_{Di} =	Az i-edik szoftver hibaforrás dezaktiválási (vagy hibaelfojtási) hatásfoka, amely a hibaforráshoz párosított hibagátló forrás (robustness source) működésének az eredménye;
Q_i =	$1 - p_{Di}$ az i-edik szoftver hibaforrás aktiválódási hatásfoka;
f_i^* =	$f_i \cdot Q_i$ az i-edik szoftver hibaforrás aktiválási intenzitása vagy másképpen: az i-edik fajta szoftver hibaintenzitás;
Z_{SWi} =	$f_i^* = f_i \cdot Q_i$ az i-edik szoftver hibaforrás aktiválásából származó szoftver hibaintenzitás (IEC-kompatibilis jelöléssel);
$f_i(A)$ =	Az i-edik szoftver hibaforrás gerjesztési intenzitása az adott A alkalmazásban;
$\vec{f}(A)$ =	$(f_1(A), f_2(A), .., f_i(A), .., f_N(A))$ alkalmazásfüggő szoftver hibagerjesztési vektor;
$\vec{Q}(R)$ =	$(Q_1, Q_2, ..., Q_i, ..., Q_N)$ szoftver hibafojtási vektor;
R =	A programcsomagba, operációs rendszerbe és a hardverbe beépített szoftver hibatűrő képesség (= szoftver robosztusság)
T(A) =	A programcsomag közepes futási ideje az A alkalmazásban;
r(A) =	$(f(A), Q(R)) \cdot T(A) = T(A) \cdot [\sum_{i=1}^{N} f_i(A) \cdot Q_i]$ a szoftverhibák várható száma az A alkalmazásban, egyetlen program lefutás alatt;
n(A) =	1/r(A) két szoftver hiba közti programfutások átlagos száma az A alkalmazásban.

8. A sokdimenziós szoftver állapottér modellje

Az előzőekben ismertetett fogalmakra alapozva lehetőség nyílik egy sokdimeziós szoftver állapottér modelljének megalkotására, amelyben

- egzakt módon áttenkinthetővé válnak a programfutás környezeti feltételei;
- a szoftver hibaforrások az állapottér meghatározott ponthalmazával azonosíthatók;
- maga a programfutás az állapottérben mozgó, ponttal jellemezhető.

A szoftver működése bonyolult számítástechnikai-műszaki környezetben, az úgynevezett működéstérben zajlik. Ezt a működésteret egy nagyon sok dimenziós állapottérrel azonosíthatjuk, amely a következő alterekkel rendelkezik:

- PCA : parancskészlet, cím- és adatmező;
- LF : a logikai feltételek összes lehetséges kombinációi, amelyek a programfutás során a döntéseket befolyásolhatják;
- IS : időskálák összessége, amelyek a hardver- és szoftver folyamatok során szinkronizmusokat, szekvenciákat, szemaforozást és maszkolást szabályoznak;
- IRK : Azon állapotok összessége, amelyek a szoftver folyamatok megszakítására vonatkozó kérések elfogadását vagy visszautasítását meghatározzák; (IRK = interrupt kérés);
- HVZ : Az összes lehetséges hardver eredetű zavaró jelenség, amely a szoftver működését befolyásolhatja;
- EMB : A rendszer kezelő személyzetének minden olyan beavatkozása, amelyet a fentebbi alterek nem tartalmaznak.

A működéstér szerkezetét egy rendkívül sokdimenziós derékszögű koordináta rendszer határozza meg, amelyben maga a működéstér diszkrét ponthálót képez, kvantált időskálákkal.

A programfutás minden egyes lépéséhez egy diszkrét állapot tartozik, amelyet a működéstérben egy pont képvisel. Az egymást követő programlépések diszkrét pontsort alkotnak. Ezt a pontsort a programfutás nyomvonalának (SPUR-jának) is nevezhetjük. Minden egyes programfuttatáshoz a működéstérben egy adott nyomvonal tartozik. Ugyanannak a programcsomagnak különböző A alkalmazásokban a nyomvonala természetszerűleg különböző, de még egy-ugyanazon alkalmazásban is az egymásutáni futtatások nyomvonala a döntési pontokban elágazhat és a bemenő paraméterek függvényében statisztikai szórást mutathat.

Ha a működésteret teljes állapottérrel sikerült modellezni, akkor a programcsomag valamennyi hibaforrása egyértelműen azonosítható az állapottérnek valamely ponthalmazával.

Ha a programfutás nyomvonala valamelyik szoftver hibaforrás (BUG) ponthalmazának egyik pontjára

r á f u t (hasonlóképpen, mint ahogy egy hajó a tenge ren aknára fut), akkor két eset lehetséges:

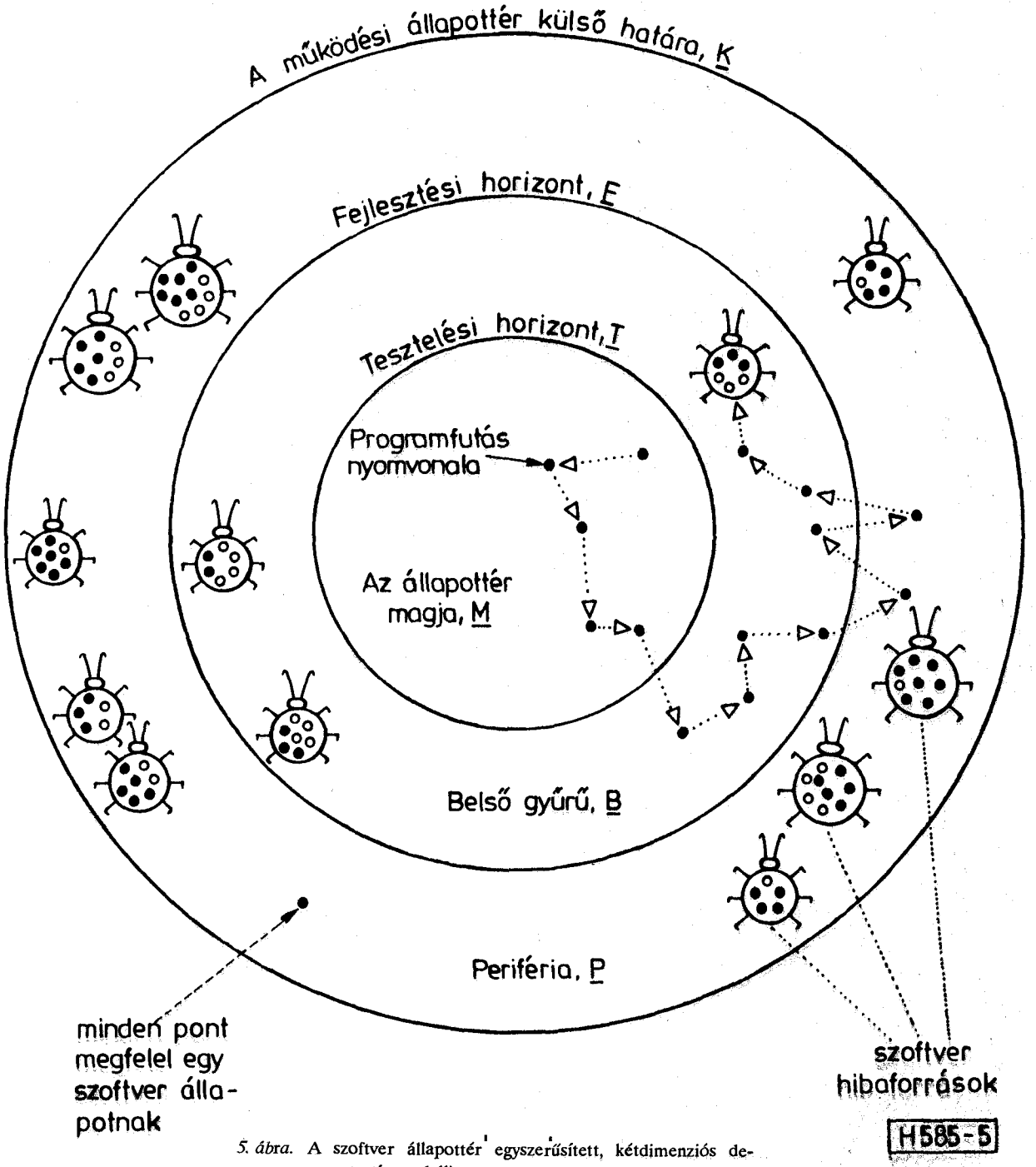
- a szoftver hibaforrás a gerjesztés nyomán aktivizálódik és szoftver hiba áll elő;
- vagy a hibaforráshoz társuló hibagátló forrás sikeresen elfojtja a gerjesztést és tovább engedi a zavartalan programfutást.

Ily módon - elméletileg - a programfutásnak és a szoftver hibaforrások aktivizálódásának igen szemléletes áttekintésére nyílik lehetőségünk. Sajnos a számítástechnika mai állása még nem teszi lehetővé több száz- vagy több ezer-dimenziós állapotterek kezelését. Ez a jövő feladata. Demonstrációs célra azonban kiválóan alkalmas az az egyszerűsített, kétdimenziós modell, amelyet az alábbiakban bemutatunk.

9. A szoftver állapottér kétdimenziós demonstratív modellje

A sok száz- vagy ezer dimenziós működési állapotteret szemléletesség kedvéért helyettesítsük a síknak egy köralakú tartományával, ahogy azt a *5. ábra* mutatja.

Az ábra szerinti szoftver állapottér megbízhatóság szempontjából három tartományra bontható.

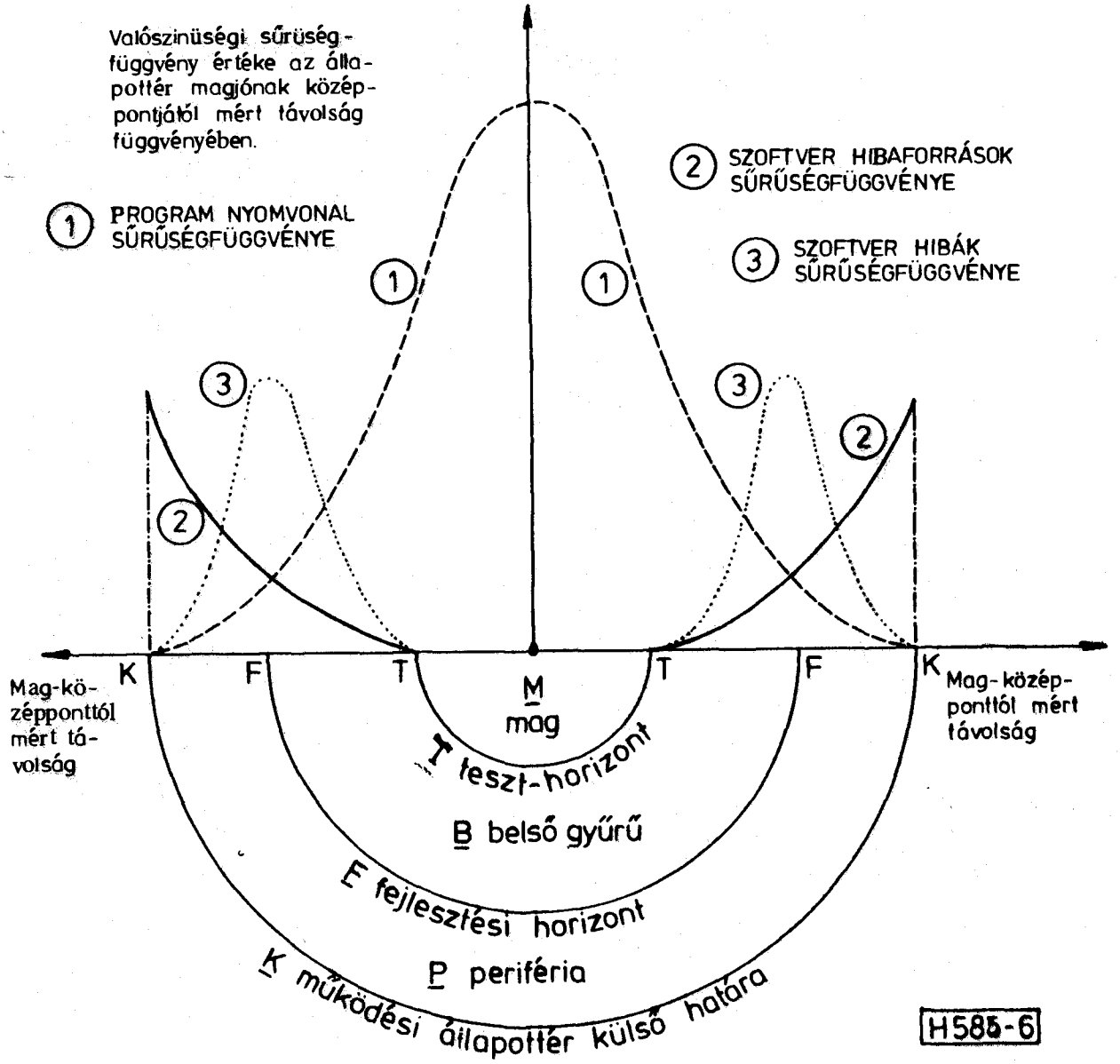


5. ábra. A szoftver állapottér egyszerűsített, kétdimenziós demonstratív modellje

Az első tartomány az állapottér magja (M), amely a T teszt-horizonton belül helyezkedik el. Ennek a tartománynak minden pontját az jellemzi, hogy 100 %-os teszt-eljárással végig vizsgálták és közben az összes szoftver hibaforrást (BUG-ot) kiküszöbölték innen. Ennek következtében a programfutás itt garantáltan hibamentes, azaz a szoftver abszolut hibamentesnek fog mutatkozni.

A második tartomány - az úgynevezett belső gyűrű (B) - a T teszt-horizont és az F fejlesztési horizont között található. A szoftver hibaforrások (BUG-ok) száma ebben a tartományban kicsi, mivel a szoftver fejlesztők a programot úgy igyekeztek megírni, hogy az előre látható hibaforrásokat kiküszöböljék. Azon kívül teszt-eljárásokkal és szimuláció segítségével a mégis bekerülő szoftver hibaforrások egy részét szintén feltárták és kiküszöbölték, jóllehet a teszt-eljárások hatásfoka ebben a tartományban már korántsem 100 %-os mélységű, mivel a működéstérnek ebben a tartományában a lehetséges állapotoknak a száma olyan hatalmas, hogy azok teljeskörű tesztelése időkorlátok miatt lehetetlen. Erre a belső gyűrűre jellemző még az is, hogy itt a rendszer hibatűrő képessége jobb, mint a gyűrűn kívül, hiszen a szoftver fejlesztők - a fejlesztési horizonton belül lévén - nemcsak az előre látható hibaforrásokat igyekeztek kiküszöbölni, hanem hibagátló források beépítésével igyekeztek a felmért lehetséges bennmaradó hibaforrások gerjesztése ellen is védelmet nyújtani. Ennek következtében kézenfekvő, hogy a bennmaradó hibaforrások dezaktiválási hatásfoka a fejlesztési horizonton innen magasabb, mint azon túl, hiszen az előre nem látható, fejlesztési horizonton túli hibaforrások aktiválódása elleni védekezés már nem lehet igazán tervszerű.

A szoftver állapottér harmadik tartománya a külső gyűrű, vagy periféria tartomány (P), amely az F fej-



6. ábra. Megbízhatósági jellemzők statisztikai eloszlása a szoftver állapottérben

lesztési horizont és a működési állapottér külső határa (K) között helyezkedik el. Ez a tartomány - több száz vagy ezer dimenzióban gondolkozva - szinte megszámlálhatatlan mennyiségű illegális működési állapotot tartalmaz, amelynek a létezéséről a szoftver fejlesztőknek nem volt tudomása, vagy legalább is létezésüket tudatosan nem vették számításba. Nyilvánvaló, hogy az ide bekerülő szoftver hibaforrások létezésére sem gondolt előre senki és azok feltárására sem készült megfelelő tesztelési eljárás. Kézenfekvő, hogy ebben a periféria tartományban a szoftver hibaforrások száma lényegesen magasabb, mint a B belső gyűrűben.

Az *5. ábra* mindezt jól szemlélteti. Kiegészítésül megjegyezzük, hogy az ábrán ninden egyes szoftver hibaforráshoz ("bogárhoz") az állapottérnek két, különböző módon jelölt ponthalmaza tartozik: az üres karikák hibaforráson belül azokat az állapotokat jelképezik, amelyekben a hiba elfojtása sikeres, míg a tömör karikák azokat, amelyekben a hiba elfojtása sikertelen.

10. Megbízhatósági jellemzők eloszlása a szoftver állapottérben

A programfutással kapcsolatos legfontosabb megbízhatósági jellemzők eloszlás-típusait a *6. ábrán* szemléltettük.

A *program* futása közben leggyakrabban az állapottér belső M magjának pontjait érinti és ritkábban kalandozik el a B belső gyűrű - még ritkábban a P periféria - területére. Statisztikai szemmel nézve úgy fogalmazhatunk, hogy a programfutás nyomvonalának valószínűségi sűrűség-függvénye az M magtartomány középpontjában veszi fel maximumát és ettől a középponttól távolodva harang-görbe szerint monoton csökken a működési állapottér külső (K) határáig, ahol értéke 0. (Lásd az 1. sorszámú görbét az ábrán).

A szoftver hibaforrások előfordulásának valószínűségi sűrűségfüggvénye az állapottér belső M magjában nulla (mivel ott szoftver hibaforrások nincsenek), majd a T tesztelési horizonttól kezdve monoton növekszik a működési állapottér K külső határáig. (2. sorszámú görbe).

Végül a hibaforrások aktiválása folytán előálló szoftver hibák előfordulásának valószínűségi sűrűségfüggvénye valahol az F fejlesztési horizont közelében veszi fel maximumát, miközben mind a T teszt-horizont, mind a K külső állpottér határ felé haladva nullához tart. (Lásd a 3. sorszámú görbét).

11. Az adott szoftver megbízhatósági modell jelentősége

Az ismertetett modell célja elsősorban az, hogy segítsen átfogó képet alkotni a szoftver megbízhatóság lényegéről és olyan rendszer-szemléletű gondolkodásmódot honosítson meg, amelynek segítségével a szoftver megbízhatóság problémái jól megragadhatók és kezelhetők. Természetesen a szoftver technika kitermelhet olyan bonyolult megbízhatósági problémákat is, amelyek lényegének megragadásához a jelenlegi modell továbbfejlesztése szükséges. Mégis remélhető, hogy az ismertetett modell - a számítástechnika gyors fejlődésével - már a közeli jövőben nem csak demostratív, hanem gyakorlati számításokra alkalmas modell lesz, amelynek segítségével számos mai szoftver megbízhatósági feladatot megoldhatunk.

IRODALOM:

[1] *Hermann Kopetz:* Softwarezuveriassigkeit. Teubner Verlag, Leipzig, 1977.

[2] *J. D. Musa:* Software Reliability Measurement. The State of the Art. EUROCON' 82 Proceedings, 655. old. Copenhagen, 1982.

[3] *B. Littlewood - J. L. Verall:* Likelihood Function of a Debugging Model of Computer Software Reliability. IEEE Transactions on Reliability, 1981. 6.

[4] *Siemens/R. Asam - N. Drenkard - H. Heinz Maier:* Qualitätsprüfung von Softwareprodukten. München, 1986.