DIPLOMAMUNKA

Svantner Viktor

Debrecen
2010

Debreceni Egyetem
Informatikai Kar

Szoftvermegbízhatóság

Témavezető:
Dr. Sztrik János
Egyetemi Tanár

Készítette:
Svantner Viktor
Programtervező matematikus

Debrecen
2010

Tartalomjegyzék

1.0 Bevezetés

A dolgozat az informatika fejlődésével egyre fontosabbá váló problémával, a szoftvermegbízhatósággal foglalkozik. Megbízhatóság alatt formálisan a szoftver egy meghatározott környezetben vett, meghatározott ideig tartó, hibamentes futásának valószínűségét értjük. A dolgozatban több megbízhatósági modellt is áttekintek, amelyeket a fejlesztők általában a projekt tesztelési szakaszának végén használnak. Ezen modellek segítségével megjósolják a várható hibák számát, ami igen hasznos a támogatási stratégia meghatározásakor.

A szoftvermegbízhatóság kiemelkedő szerepét az is bizonyítja, hogy már 1991 – ben az ANSI/IEEE a STD-729-1991 – es szabványban foglalkozott a témával.

Habár az idővel kapcsolatos valószínűségi függvényként definiálják, a szoftver megbízhatósága eltérő a hardver megbízhatóságától, hiszen azzal ellentétben nem direkt függvénye az időnek. Az elektronikus és mechanikus részek öregednek és elhasználódnak idővel, a szoftver azonban nem „rozsdásodik” az életciklusa során, nem változik, hacsak nincs benne szándékos változtatás, frissítés.

Elméletben egy gondosan felépített, megfelelő mértékben tesztelt szoftver soha nem hibásodik meg, hiszen a program nem öregszik, a számítógép pedig, amelyen fut, folyamatosan biztosítja a kellő erőforrásokat. A gyakorlatban azonban ez másképpen alakul:

1, 1982 – ben a Falkland – szigeteki háborúban az angolok Sheffield rombolóját azért süllyesztette el az argentinok egyik Exocet (angol gyártmányú) rakétája, mert a Sheffield radarrendszere a rakétát tévesen „barátiként” azonosította.

2, 1991. február 25 – én az öbölháborúban egy amerikai Patriot ellenrakéta irányítórendszerének nem megfelelő működése okozott szerencsétlenséget. A rendszer az idő kiszámításához hasznát egy 1/10 – el való szorzást. Az 1/10 azonban számítógépes aritmetikában végtelen tizedestört: $\frac{1}{10} = \left(\frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \ldots\right)$. Ennek a tárolására egy 24 bites fix pontos regisztert használtak. Így a végtelen tizedestört 24. bitje utáni részt a rendszer levágta. Ez másodpercenként 0.000000095 másodperces eltérést eredményezett, ami a rendszer 100 órás működése után 0.34 másodpercre növekedett.

A beérkező iraki Scud lövedék másodpercenkénti 1.676 méteres sebességgel közeledett, vagyis a 0.34 szekundumos eltérés alatt kb. 570 métert tett meg. Ennek következtében a rakéta elhárító rendszer képtelen volt befogni és eltalálni a beérkező lövedéket. Az eredmény 28 halott és több mint 100 sérült.

Látható tehát, hogy a szoftverek nem megfelelő működése valós probléma, amelytől akár emberek élete is függhet.

A dolgozathoz mellékelek egy analitikus programot, amely segítségével az áttekintett modellekben számíthatunk megbízhatóságot, így a szoftverek széles körét tudjuk vele analizálni.

2.0 A szoftvermegbízhatóság modellezésének elemei

A hardverek illetve a szoftver eszközök működőképessége eltérő. A hardvereké egyedileg változik, véletlen tényezőktől függ, mint például a környezeti hatások, vagy az elhasználódás mértéke.

Ezzel ellentétben a szoftverek példányai azonosak, és eltekintve az adathordózók esetleges meghibásodásaitól, időben állandóak. A program részeit két csoportban sorolhatjuk. Az egyik csoportot a megfelelően működő programrészek alkotják, a másikat pedig a hibásan működőek. Az időbeli állandóság miatt a hibátlan programrész mindig hibátlanul, a hibás viszont mindig hibásan működik.

Így arra a következtetésre juthatunk, hogy a szoftver megbízhatóságának statisztikai jellegét nem a belső tulajdonságok határozzák meg, hanem az, hogy milyen gyakorisággal lépünk be a hibás programrészbe.

Emiatt előfordulhat, hogy egy hibákkal teli program használata esetén a felhasználó semmit sem vesz észre, mert a program soha nem lép be valamely hibás részébe. Ennek tipikus példája a Pentium processzorok piacra dobásakor felfedezett programhiba, amelyre egy matematikatanár világított rá. A probléma egy számelméleti algoritmus hibás működése volt, amelyet a gyári teszteléskor senki sem ellenőrzött.

A felhasználó szempontjából ez a jelenség nem mérvadó, hiszen őt a teljes rendszer hibátlan működése érdekli, gyakran nem is tud különbséget tenni a hardver és a szoftver hibái között. Egy tipikus példája ennek, az adathordozó meghibásodása miatt válik működésképtelenné a szoftver.

A szoftver esetében a hibás működés oka majdnem mindig a nem megfelelő tervezésben keresendő. Semmilyen fizikai folyamat alapján nem jósolható, hiszen a tervezés emberi tényezőitől függ. Az egyetlen javítási lehetőség az újratervezés, az újraprogramozás.

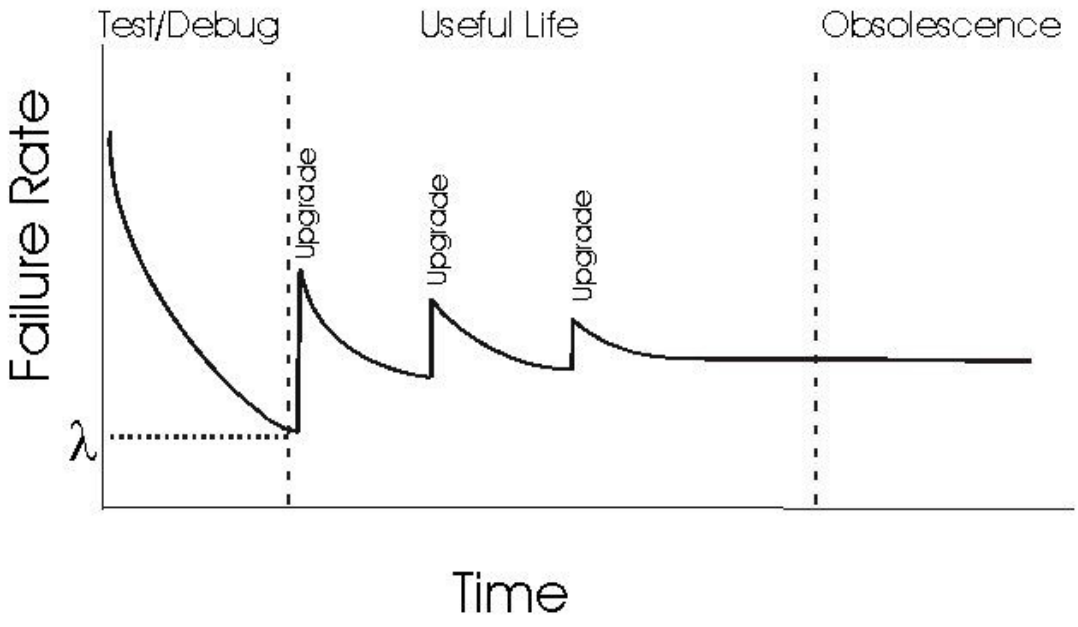
A megbízhatóság nem növelhető hasonló tartalékegységes alkalmazásával, hiszen a hibás programrész ott van mindegyik példányban, így a helytelen működés valószínűsége nem csökken a tartalékok alkalmazásával. Csak akkor lehet ilyen módon csökkenteni a szoftverhibák valószínűségét, ha a párhuzamos ágakban nem ugyanazon példányok vannak, hanem más fejlesztők által elkészített, és így eltérő programok.

A hibák általában nem jelezhetők előre az egyes programrészekre vonatkozó megállapításokból, hiszen a hibák véletlenszerűen helyezkednek el a programban, ezáltal bármelyik programegység lehet hibás.

2.1 Meghibásodási intenzitás

Ha megfigyeljük a szoftverhibákat az idő előrehaladtával, megkapjuk az szoftver meghibásodási intenzitását. A tesztelési szakaszban igen sok hibára derül fény, míg a szoftver életciklusának végén, az elavulási szakaszban már szinte sosem kerül sor hibás működésre. Az életciklus ezen két része között, az aktív használat közben a hibák gyakoriságának görbéje fűrészfog alakot mutat. Ezt a szoftver időbeli állandóságával lehet magyarázni, hiszen a programban nem keletkeznek új hibák, amennyiben nem módosítjuk azt. Tehát csak módosítás során keletkezhetnek, azaz frissítések során.

A következő ábra szemlélteti a szoftver meghibásodási rátáját:



2.2 Hibák számlálása kontra meghibásodási intenzitás

A legtöbb megbízhatósági modell tartalmaz egy mérőszámot, amely a kezdeti hibák számát reprezentálja. Amennyiben feltételezzük, hogy a szoftverben lévő hibák száma véges, úgy a fennmaradó hibák számának becslése kerül a figyelem középpontjába.

A hibák száma azonban nem minden esetben áll arányban a szoftver megbízhatóságával. Sokkal megbízhatóbb egy lényegesen több hibával rendelkező program, amelyben a hibák relatíve ritkán fordulnak elő, mint egy olyan program, amely kevés, de gyakran felbukkanó hibával rendelkezik. Ezért sokkal fontosabb a meghibásodási intenzitás mérése.

A szoftvermegbízhatósági modellek, melyek főleg a tesztelési szakaszban alkalmazhatók, szintén modelljei egy javítható rendszernek. Tehát a szoftverrendszerekben létezik a hibajavítás folyamata. A meghibásodási intenzitása egy javítható rendszernek sokkal hasznosabb, mint pusztán a hibák száma. Ezért lehetséges a hardvermegbízhatóságban már bizonyított nem-homogén Poisson-folyamat modellek használata szoftverek vonzatában.

A meghibásodási intenzitás becslése általában nem egyszerű feladat, azonban több egyszerű dinamikus modellben is elérhető egy egyszerű becslés. Az intenzitás több dologtól is függ, mint például a tesztelés intenzitása, vagy homogenitása, és célszerű olyan modellt választani, amelyek képesek ezeket kezelni.

2.3 A hiba nagysága

A modellek nagy része, különösképpen a régebbiek azt tételezték fel, hogy a hibák nagysága azonos. Ebből az is következik, hogy bármely hibát is javítják ki, a szoftver megbízhatósága növekedni fog, méghozzá ugyanakkora mértékben. Ez egy igen széles körben vitatott kérdés, mert általában a gyakorlatban nem igaz.

Ennek ellenére nagyon nehéz a hibák nagyságát jól definiálni. Általában a hibák nagyságát az előfordulásuk valószínűségével szokták megadni. Ez azonban nem mérhető, a megbízhatósági modellek éppen ezt próbálják meg becsülni a tesztelési szakaszban. Más megközelítésben a hiba nagysága lehet az általa érintett sorok száma a kódban, vagy a hibát kiváltó bemenő adatok mennyisége. Emellett a hiba nagysága definiálható a súlyosságával is.

Amennyiben egyforma nagyságúnak tekintjük a hibákat, a fennmaradó hibák becslése általában optimista, hiszen először a gyakrabban előforduló hibákat fedezik fel, a rejtettebbeket sokkal több erőfeszítésbe kerül.

2.4 Véletlenszerű, homogén tesztelés

A szoftverek tesztelése elméletileg a lehetséges bemenő adatok közül véletlenszerűen válogatva, homogén módszerrel zajlik. A gyakorlatban azonban nem homogén a tesztelés, hiszen számos körülmény: betegség, tesztkörnyezet elérhetősége befolyásolja. A bemenő adatok sem véletlenszerűek, speciális adatokat használnak a hatékonyabb tesztelés érdekében.

A nem-homogén tesztelés érdekében figyelembe kell vennünk, hogy szoftvernek a kibocsátása utáni megbízhatóságát szeretnénk mérni. Ezért célszerű a mindennapi használatához leginkább hasonlító adatokkal tesztelni.

3.0 Nem-homogén Poisson-folyamat modellek

3.1 Alap Nem-homogén Poisson-Folyamat modell

A jól felépített, sztochasztikus folyamatokat alkalmazó modellek általános osztályát, a nem-homogén Poisson-folyamat modelleket, már régóta használják a megbízhatóság elemzésében, igen nagy sikerrel. Akkor igazán hasznosak, ha olyan meghibásodási folyamatokat kell leírni, amelyek valamilyen trendet követnek, mint például a növekedési modellek.

A modell lényege, hogy a *t* időpillanatig bekövetkező hibák száma: N(t) egy nem-homogén Poisson-folyamat segítségével írható le. Tehát az N(t) egy m(t) paraméterű Poisson eloszlású valószínűségi változó. Ekkor annak a valószínűsége, hogy az N(t) éppen az *n* értéket veszi fel:

$$P\{N(t)=n\} = \frac{[m(t)]^n}{n!} e^{-m(t)}.$$

Az m(t) függvény a [0, t) időintervallumban bekövetkezett hibák számának várható értékét írja le, hiszen a Poisson eloszlás várható értéke maga a paraméter. Ezért az m(t) nagyon jó leíró mennyisége a meghibásodásnak.

A nem-homogén Poisson-folyamatra az alábbi tulajdonságoknak teljesülniük kell:

1, N(0) = 0,

2, { N(t) $t \geq 0$ } független növekményekkel rendelkezik,

3, P{ N(t+h) – N(t) = 1 } = $\lambda(t) + o(h)$,

4, P{ N(t+h) - $N(t) \geq 2$ } = o(h).

A fent szereplő $\lambda(t)$ függvény, amit pillanatnyi meghibásodási intenzitásnak nevezünk, a következőképpen definiálható:

$$\lambda(t) = \lim_{\Delta t \to 0_+} \frac{P\{N(t+\Delta t) - N(t) > 0\}}{\Delta t}.$$

Adott $\lambda(t)$ mellett az m(t) teljesíti a

$$m(t) = \int_0^t \lambda(s)ds - t.$$

Ezt megfordítva, ha tudjuk m(t) -t, akkor a *t* időpillanatban vett meghibásodási intenzitás kifejezhető a következőképpen:

$$\lambda(t) = \frac{dm(t)}{dt} = \int m(t)\,dt.$$

Általánosságban, ha különböző nem csökkenő m(t) függvényeket alkalmazunk, különböző nem-homogén Poisson-folyamat modelleket kapunk. A legegyszerűbb esetben amikor a $\lambda(t)$ egy konstans, a nem-homogén Poisson-folyamat modell lényegében egy homogén Poisson-folyamat modell lesz, amelyben a paraméter *t* – nek valamely skalárszorosa.

Annak köszönhetően, hogy a paraméter nagyon széles skálán mozoghat, az idők során nagyon sok nem-homogén Poisson-folyamat modellt állítottak fel, és tanulmányoztak.

Az első ilyen modellt Schneidewing ajánlotta 1975 – ben. Az első közismert nem homogén Poisson-folyamat modellt Goel és Okamoto mutatta be 1979 – ben, amit később tovább általánosítottak, és más szerzők módosították is, hogy még hatékonyabb modelleket állítsanak elő.

Ha a szoftverben maradt további hibákat $\bar{N}(t)$ szeretnénk megadni a *t* időpillanatban, akkor a következő egyenletet kapjuk:

$$\bar{N}(t) = N(\infty) - N(t),$$

ahol $N(\infty)$ azon hibák száma, amelyeket végtelen ideig tartó tesztelés során fedezünk fel, vagyis az összes hiba darabszáma, hiszen feltételeztük, hogy a szoftverben véges sok hiba fordul elő.

Ez szintén a nem-homogén Poisson-folyamat elméletét követi, mivel $\bar{N}(t)$ Poisson eloszlású, $m(\infty) - m(t)$ paraméterrel. Ennek megfelelően Annak a valószínűsége, hogy $\bar{N}(t)$ éppen egy *k* konstanssal egyenlő:

$$P\{\bar{N}(t) = k\} = \frac{[m(\infty) - m(t)]^k}{k!} e^{-(m(\infty) - m(t))}.$$

Általában az m(t) több ismeretlen paramétert tartalmaz. Ezeknek a meghatározása általában a Maximum Likelihood vagy a legkisebb négyzetek módszerével történik.

Jelölje n_i azon hibák számát, amelyeket az $[s_{i-1}, s_i)$ időintervallumban fedeztünk fel, ahol $0 = s_0 \le s_1 \le s_2 \le ... \le s_n,$ és $s_i \; i \ge 0,$ a szoftver futási ideje a teszt elindítása óta.

Az m(t) paraméterű nem-homogén Poisson-folyamat modell valószínűségi függvénye ekkor a következő:

$$L(n_1, n_2, \ldots, n_k) = \prod_{i=1}^{k} \frac{[m(s_{i-1}) - m(s_i)]^{n_i}}{n_i!} e^{-(m(s_{i-1}) - m(s_i))}.$$

Az m(t) – beli paramétereket ezen valószínűségi függvény maximalizálásával becsülhetjük. Erre általában numerikus módszereket alkalmaznak.

A nem-homogén Poisson-folyamatoknak sok előnyös tulajdonságuk van. Az egyik az, hogy a Poisson eloszlás miatt az nem-homogén Poisson-folyamatok összege is nem homogén Poisson-folyamat, így a különböző folyamatokból vett meghibásodási adatokat összemixelhetjük és az eredmény szintén nem-homogén Poisson-folyamat lesz, amelynek paramétere a részparaméterek összege lesz.

A másik jó tulajdonsága a nem-homogén Poisson-folyamatoknak, hogy könnyen lehet belőlük homogén Poisson-folyamat modelleket készíteni. Ezt úgy tehetjük meg, hogy az m(t) paraméter helyett egy m(a(t)) paramétert használunk, hiszen ha N(t) egy nem-homogén Poisson-folyamat m(t) paraméterrel, akkor $N^*(t) = N(a(t))$ is nem-homogén Poisson-folyamat, amelynek paramétere m(a(t)). Amennyiben az a(t) – t éppen $m^{-1}(t)$ - nek választjuk, akkor az N(t) paramétere *t* lesz, ami egy konstans.

3.2 Goel – Okumoto (GO) modell

1979 – ben Goel és Okumoto egy egyszerű modellt mutatott be a szoftver meghibásodási folyamat leírására. Feltételezték, hogy a meghibásodási folyamat nem-homogén Poisson-folyamat egyszerű paraméterfüggvénnyel.

A modellre vonatkozó általános előfeltételek a következők voltak:

(1) A t. időpillanatig felfedezett hibák száma Poisson eloszlású

(2) A hibák egymástól függetlenek, és ugyanakkora valószínűséggel találják meg őket

(3) A megtalált hibákat azonnal kijavítják, és a szoftverben nem keletkeznek új hibák

Ebben a modellben a meghibásodási folyamat egy olyan nem-homogén Poisson-folyamat, melynek a paraméterfüggvénye:

$$m(t) = a\left(1 - e^{-bt}\right), \quad a > 0, b > 0;$$

ahol a és b paraméterek, amelyeket a korábbi tesztelésből fakadó, már meglévő hibaadatok határoznak meg.

Ebben a modellben $m(\infty) = a$ és m(0) = 0. Mivel $m(\infty)$ a szoftverben megtalálható hibák száma, így az a paraméter a tesztelési szakaszban maximálisan megtalálható hibák száma lesz. A b paramétert pedig az egyes hibák felbukkanási valószínűségét jelenti.

Az intenzitásfüggvény, amit az m(t) deriváltjaként definiálunk a következőképpen néz ki:

$$\lambda(t) = abe^{-bt}.$$

A fennmaradó hibák számának várható értéke a t. időpillanatban:

$$E\,\bar{N}(t) = E\left[N(\infty) - N(t)\right] = m(\infty) - m(t) = a - a(1 - e^{-bt}) = ae^{-bt}.$$

3.3 Musa végrehajtási idő alapú modellje

1975 – ben Musa egy olyan elméletét mutatta be a szoftvermegbízhatóságnak, amely a végrehajtási időn alapult. Mivel az egyes tesztek végrehajtási ideje sokkal közelebb áll a valós tesztelési erőfeszítések kifejezéséhez, így az sokkal jobb mérőszáma az időnek, mint a naptári idő.

Jelölje N_0 *és* $\mu(\iota)$ a kezdeti hibák számát, valamint a kijavított hibák számát ι tesztelési idő elteltével, amikor is a tesztelési időt végrehajtási időben adjuk meg.

Ekkor a szoftverben megmaradó hibák száma: $\lambda(\iota) = fK\left[N_0 - \mu(\iota)\right]$, ahol f és K a tesztelési fázissal kapcsolatos paraméterek. Az f mennyiség tekinthető úgy, mint egy lineáris végrehajtási gyakoriság, amelyet úgy kapunk, hogy az átlagos utasítás végrehajtási rátát elosztjuk a programban lévő utasítások számával.

A hibák kijavításának intenzitása a hibák megjelenési intenzitásával arányos:

$$\frac{d\,\mu(\iota)}{\iota} = BC\lambda(\iota),$$

ahol B a hiba redukálásának faktora, a C – t pedig annak az erőfeszítésnek a nagysága, amit a tesztelés során a hibák felkutatására fordítunk.

A fentebbi két egyenletből a következőt kapjuk:

$$\frac{d\,\mu(\iota)}{\iota} = BC\,\lambda(\iota) = BCfK\left[N_0 - \mu(\iota)\right].$$

Ez egy differenciálegyenlet, amelyek a következőképpen lehet átírni: $\mu'(\iota) + BCfK\,\mu(\iota) - BCfKN_0 = 0.$

Amennyiben a kezdeti feltételek a $\mu(\iota) = 0,$ ha $\iota = 0,$ akkor az egyenlet megoldása:

$$\mu(\iota) = N_0\left(1 - e^{-BCfK\,\iota}\right).$$

Ez a függvény a Musa modell paraméterfüggvénye.

Amennyiben a felfedezett hibák száma helyett, a meghibásodási folyamatot a ι időben felfedezett hibák számának összegével $\nu(\iota)$ definiáljuk, akkor hasonló következtetésre juthatunk, mint az előző esetben. A B hibaredukálási faktort használva:

$$\nu(\iota) = \frac{N_0}{B}\left(1 - e^{-BCfK\,\iota}\right).$$

4.0 Növekedési modellek

A megbízhatóságot gyakran azzal a valószínűséggel azonosítják, hogy egy adott rendszer meghatározott működési feltételek mellett, egy meghatározott ideig hibátlanul működik. Ezáltal a megbízhatóság a hibák között eltelt idővel vagy annak reciprokával, a meghibásodási intenzitással van szoros összefüggéseben.

A tesztelési időszakban az egységnyi tesztelési idő alatt felfedezett hibák száma adja a hibák felfedezésének intenzitását. A tesztelés előrehaladtával egyre kevesebb hiba kerül felszínre egységnyi idő alatt, amikor pedig ez az arány egy bizonyos határ alatt van, a szoftver készen áll a kibocsátásra. A tesztelés során az idővel felfedezett hibák számát ábrázolhatjuk egy görbe segítségével, amely szigorúan monoton nő, de egyre kisebb mértékben.

A szoftvermegbízhatóság növekedési modelljei matematikai függvények segítségével próbálják meg statisztikailag közelíteni ezt a görbét. Ezeket a függvényeket arra használják, hogy megbecsüljék a jövőbeli meghibásodási intenzitást, valamint becslést adnak a kódban később felfedezett hibák számáról is. Ez segíti a fejlesztőket a kibocsátási idő valamint a későbbi karbantartás tervezésében.

Napjainkban több ilyen modell is létezik, azonban nincs univerzális, mindegyiknek van előnye, és hátránya, így több modell kipróbálása után lehet megmondani, hogy az adott környezethez mi illik a legjobban.

Az azonban mindegyikben közös, hogy a hibák száma véges, a teszteléshez pedig akár végtelen idő is rendelkezésre állhat.

4.1 Duane modellje

A Duane modellben, amelyet Weibull folyamat modellként is neveznek a paraméterfüggvény a következő:

$$m(t) = \left(\frac{t}{\alpha}\right)^{\beta}, \quad \alpha > 0,\ \beta > 0.$$

Az α és β paraméterek, amelyeket már begyűjtött meghibásodási adatok alapján lehet becsülni.

A hibák megjelenésének intenzitása a t időpillanatban:

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1}, \quad \alpha > 0, \beta > 0.$$

Az egyik legfontosabb előnye ennek a modellnek, hogy ha ábrázoljuk a megjelent hibák számát, valamint a teljes tesztelési időt, akkor a kapott pontok egy egyeneshez közeli alakot formálnak, ha helyes a modell.

Ezt abból is lehet látni, hogy az m(t) és t közötti reláció átírható a következő alakra:

$$\ln m(t) = \beta \ln\left(\frac{t}{\alpha}\right) = \beta \ln t - \beta \ln \alpha = a + b \ln t,$$

amennyiben $a = -\beta \ln \alpha$ és $b = \beta$.

Mint látható ln(m(t)) lineáris függvénye ln(t) - nek, és ennek köszönhetően az α és β paraméterek grafikusan becsülhetőek, így a modell helyessége könnyen ellenőrizhető.

Duane modelljének egyik hátránya, hogy a 0. időpillanatban végtelen nagyságú intenzitást ad eredményül. Ezt azonban később Littlewood – nak sikerült megoldania.

1984 – ben Littlewood bemutatta a Módosított Duane modellt, amelyben módosította a paraméterfüggvényt, a következő módon:

$$m(t) = k\left[1 - \left(\frac{\alpha}{\alpha + t}\right)^{\beta}\right], \quad \alpha > 0, \beta > 0, k > 0.$$

A *k* paraméter a pillanatnyilag detektált hibák száma.

Ebben az esetben a hibák megjelenésének intenzitása:

$$\lambda(t) = m'(t) = k\,\beta\,\alpha^{\beta}(\alpha + t)^{-1-\beta}.$$

Ekkor a 0. időpillanatban a meghibásodási intenzitás $\frac{k\,\beta}{\alpha}$, ami egy véges érték, és az intenzitás tart 0 – hoz, ahogyan a t – vel tartunk a végtelenbe.

4.2 Alap Musa model

Ezt a modellt alapmodellnek, az angol terminológiában „Basic” - nek nevezik, mert egy egyszerű megközelítését adja a meghibásodási intenzitásnak. Ebben a modellben, a rendszer hibák közötti működési ideje exponenciális eloszlású. Így a meghibásodási intenzitást a következőképpen kapjuk:

$$\lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{\nu_0}\right), \quad \text{ahol}$$

$\lambda(\mu)$ a meghibásodási intenzitás

λ_0 meghibásodási intenzitás a szoftver üzembehelyezésekor

μ a várható hibák száma egy megadott időintervallumban

ν_0 az összes megjelenő hibának a száma, ha az idővel tartunk a végtelenbe

A megadott formulából látszik, hogy ennek a modellnek a megfelelő használatához szükségesek létező meghibásodási adatok. Ezek az adatok csak egy már futó rendszerből nyerhetők ki pl. egy béta tesztelés során.

Az Alap Musa modellt ajánlják az ipari projektek nagy részéhez, hiszen számos különböző projektben bizonyított már, és elérte a tervezője által megfogalmazott célt: széles körben alkalmazható, egyszerű és hasznos mennyiségek számítását végzi.

4.3 További növekedési modellek

Mivel a szoftvermegbízhatóságot tekinthetjük növekedési folyamatként, így a közelítésére használhatók a hardvermegbízhatóságban alkalmazott növekedési modellek, valamint a gazdasági és népességi növekedést előrejelző modellek is.

A Gompertz növekedés modellben a paraméterfüggvény:

$$m(t) = ka^{b^t}, \quad 0 < a, b < 1, k > 0.$$

Mivel b < 1, így ha t – vel tartunk a végtelenbe: $m(\infty) = ka^0 = k$. Ezért ebben a modellben a k jelöli a szoftverben már a kezdetekkor meglévő hibák várható értékét, valamint *a* és *b* paraméterek, amelyeket a már meglévő meghibásodási adatokból származtatunk.

5.0 Többállapotú „*k* az *n*-ből” rendszerek modellezése, és megbízhatóságuk számítása

5.1 Bináris „*k* az *n*-ből” rendszerek

Egy *n* komponensből álló rendszert bináris „*k* az *n*-ből: G” rendszernek nevezzük, amennyiben a rendszer működik, ha legalább *k* db komponense működik. Egy *n* komponensből álló rendszert bináris „*k* az *n*-ből: F” rendszernek nevezzük, amennyiben a rendszer meghibásodik, ha legalább *k* db komponense hibásodik meg.

Megadható a megbízhatóságot hatékonyan számító algoritmus független komponensekből álló bináris „*k* az *n*-ből” rendszerek számára, a következőképpen:

$$R(n,k) = p_n * R(n-1,k-1) + q_n * R(n-1,k),$$

ahol R(n, k) egy rekurzív függvény, amely egy „*k* az *n*-ből: G” rendszer megbízhatóságát reprezentálja, p_n az n komponens megbízhatósága, valamint $q_n = 1 - p_n$. A peremfeltételek pedig:

$$R(n, 0) = 1,$$
$$R(n, k) = 0, \text{ ha } 0 < n < k.$$

5.1.1 Súlyozott bináris „*k* az *n*-ből” rendszerek

Súlyozott „*k* az *n*-ből” rendszernek nevezzük azokat a rendszereket, amelyekben az *i* komponenshez a w_i, $w_i > 0$, $i = 1,2,\ldots,n$ súly tartozik. A komponensek súlyainak összege w, $w = \sum_{i=1}^{n} w_i$. A rendszer akkor és csak akkor működik, ha a működő komponensekhez tartozó súlyok összege legalább egy előre definiált *k* érték.

A komponensekhez tartozó súlyok annál nagyobbak, minél fontosabb az adott komponens a rendszer számára. Szintén rekurzív formulával adható meg annak a valószínűsége, hogy egy *j* komponensből álló rendszer működő komponenseihez tartozó súlyok összege minimum *i*.

Ekkor az R (k, n) a megbízhatósága a súlyozott „*k* az *n*-ből: G” rendszernek. A következő rekurzív formula használható az ilyen rendszerek megbízhatóságának kiszámításához:

$$R(i, j) = p_j R(i - u_j, j - 1) + q_j R(i, j - 1),$$

a peremfeltételek pedig:

$$R(i, j) = 1, \text{ ha } i \le 0, \; j \ge 0,$$

$$R(i, 0) = 0, \text{ ha } i > 0.$$

5.2 Többállapotú rendszerek

A gyakorlatban több olyan komponens és rendszer is található, amely több mint két különböző szintű teljesítményre képes.

Tekintsünk példaként egy erőműben lévő áramfejlesztőt, amely teljes kapacitással működik a rendszer hibátlan működése esetében. Ekkor a rendszerben bekövetkező különféle hibák esetében a teljesítmény különböző módon csökken. Vannak olyan hibák, amelyek az áramfejlesztő teljes leállásához vezet, de léteznek olyanok is, amelyek csak redukálják a teljesítményt, különböző mértékben. Így annak függvényében, hogy az áramfejlesztő milyen teljesítmény leadására képes, a rendszer különböző állapotokban lehet.

5.3 Többállapotú „*k* az *n*-ből” rendszerek modellezése

A többállapotú „*k* az *n*-ből” rendszerek leírására több modell is született. Az első ilyen modell El-Neweihi nevéhez fűződik. Az ő modelljében a rendszer állapota a k. legjobb komponens állapotaként volt definiálva.

Az általános többállapotú „*k* az *n*-ből” rendszer modelljében, amely Huang nevéhez fűződik, a rendszerben *k* db különböző érték lehet, amelyek az egyes állapotokat jellemzik. Ennek a modellnek egyik hátránya azonban, hogy kevés gyakorlati alkalmazás illeszthető bele.

A következő részben mélyebb betekintést nyújtok a különböző modellekbe. Ehhez szükséges egy jelölésrendszer valamit kezdeti feltételek, amelyek a következők:

Kezdeti feltételek:

1, Az egyes komponensek és a rendszer állapottere: {0, 1, 2, …, M}

2, A rendszer állapotát teljes mértékben meghatározza a komponensei állapota.

Jelölésrendszer:

n:	A rendszer komponenseinek száma
M:	A legmagasabb állapotszint, a rendszerre és a komponenseire nézve
x_i:	Az *i* komponens állapota. $x_i = j$, Ha *i* a *j* állapotban van $0 \le j \le M$, $1 \le i \le n$.
x:	egy n dimenziós vektor, ami a komponensek állapotait reprezentálja. $x = (x_1, x_2, ..., x_n)$.
$\phi(x)$:	A rendszer állapota, $0 \le \phi(x) \le M$.
k_j:	A j szinthez tartozó k érték, az általános többállapotú „*k* az *n*-ből” rendszerre vonatkozóan.
$P_{s,j}$:	$P(\phi(x) \ge j)$
$Q_{s,j}$:	$P(\phi(x) < j)$
$r_{s,j}$:	$P(\phi(x) = j)$
P(•):	Rekurzív függvény a „*k* az *n*-ből: G” rendszerre vonatkozóan.
Q(•):	Rekurzív függvény a „*k* az *n*-ből: F” rendszerre vonatkozóan.
k:	A **k** vektor a többállapotú „*k* az *n*-ből: G” rendszerre vonatkozóan $k = (k_1, k_2, ..., k_M)$.
P:	A komponensek állapotának eloszlási mátrixa a nominális többállapotú „*k* az *n*-ből” rendszerre vonatkozóan.
$p_{n,j}$:	Annak a valószínűsége, hogy az *n* komponens a *j* állapotban van.
k^j:	A generált **k** vektor, amikor az *n* komponens a *j* állapotban van.
P^j:	A generált **P** mátrix, amikor az *n* komponens a *j* állapotban van.

5.4 Többállapotú „*k* az *n*-ből” rendszermodellek

5.4.1 Huang modellje

Ez volt az első modell, amely megengedte, hogy a különböző állapotokhoz különböző *k* értékek társuljanak.

Egy n komponensből álló rendszert általánosított többállapotú „*k* az *n*-ből: G” rendszernek nevezünk, amennyiben hogyha $\phi(x) \geq j \quad (1 \leq j \leq M)$ akkor létezik egy olyan egész l $(j \leq l \leq M)$ érték, hogy legalább k_l komponens van *l*, vagy afölötti állapotban.

Amennyiben fennáll, hogy $k\,1 \leq k_2 \leq ... \leq k_M$, akkor a rendszert növekvő többállapotú „*k* az *n*-ből: G” rendszernek nevezünk.

Ha $k\,1 \geq k_2 \geq ... \geq k_M$, akkor a rendszert csökkenő többállapotú „*k* az *n*-ből: G” rendszernek nevezünk.

Amikor $k\,1 = k_2 = ... = k_M = k$, azaz k_j konstans, akkor a rendszert konstans *k* az *n*-ből: G” rendszernek nevezünk.

Hasonlóképpen definiálhatjuk a többállapotú *k* az *n*-ből: F” rendszereket is. Egy n komponensből álló rendszert általánosított többállapotú „*k* az *n*-ből: F” rendszernek nevezünk, amennyiben hogyha $\phi(x) < j \quad (1 \leq j \leq M)$ akkor legalább k_l komponens állapota *l*, vagy *l* alatti állapot, minden *l* – re, melyre $j \leq l \leq M$.

A következő példán keresztül megfigyelhetjük ennek a modellnek a gyakorlati alkalmazását. Tekintsünk egy Oracle RAC (Real Application Clusters) környezetben működő adatbáziskezelő-rendszert, amelynek három példánya fut, melyek egyenként 1000 felhasználót tudnak kiszolgálni. Mindegyik példány lehet a 0, 1 vagy 2 állapotok valamelyikében attól függően, hogy hibátlanul működik, valamilyen hiba folytán csak kisebb teljesítménnyel képes üzemelni, esetleg kritikus hiba miatt működésképtelen.

Amikor egy komponens a 2 – es állapotban van, 1000 felhasználó kiszolgálására képes, az 1 – es állapotban 200, míg a 0 – ás állapotban 0 felhasználó kiszolgálására képes. Ennek megfelelően a rendszer is több állapotban lehet. A rendszer a 2 – es állapotban van, ha minimum 1000 felhasználót képes kiszolgálni, 1 – es állapotban van, ha minimum 400 – at, és 0 – ás állapotban van egyébként.

Ezek alapján a rendszert kezelhetjük egy csökkenő többállapotú „*k* az *n*-ből: G” rendszerként. Ekkor n = 3, M = 2, $k_1 = 2$, $k_2 = 1$. A rendszer tehát a 2 – es állapotban van, amennyiben minimum egy komponense a 2 – es állapotában van, 1 – es állapotban van a rendszer, ha minimum egy komponense van a 2 – es, vagy két komponense van az 1 – es állapotában, vagy fölötte. Egyébként a rendszer a 0 – s állapotában van.

5.4.2 Tian modellje

Tian a többállapotú „*k* az *n*-ből: G” rendszert a következőképpen definiálta:

Egy n komponensből álló rendszert többállapotú „*k* az *n*-ből: G” rendszernek nevezünk, ha $\phi(x) \geq j \quad (1 \leq j \leq M)$ valahányszor legalább k_l komponens van l, vagy afölötti állapotban, minden olyan l – re, melyre $1 \leq l \leq j$.

Ez alapján egy rendszer akkor van a *j*, vagy afölötti állapotban, ha az egyes állapotokban lévő komponensek száma megfelelő az összes 1 – től *j* – ig terjedő állapotban. Ezzel szemben Huang modelljében a rendszer a *j* vagy afölötti állapotban volt, ha bármely j – től M – ig terjedő szinten megfelelő számú komponens volt.

Létezik egy speciális esete az itt bemutatott többállapotú „*k* az *n*-ből: G” modellnek, ez pedig a csökkenő többállapotú „*k* az *n*-ből: G” rendszer, amelyhez szükséges és elégséges feltétel, hogy $k1 \geq k_2 \geq ... \geq k_M$.

Ebben a modellben egy n komponensből álló rendszert többállapotú „*k* az *n*-ből: F” rendszernek nevezünk, ha létezik egy olyan l, $(1 \leq l \leq j)$ egész szám, hogy $\phi(x) < j \quad (1 \leq j \leq M)$ valahányszor legalább k_l komponens van az l szint alatti állapotban.

Minden többállapotú „*k* az *n*-ből: F” rendszerhez létezik vele ekvivalens többállapotú „*k* az *n*-ből: G” rendszer, és fordítva. A többállapotú „*k* az *n*-ből: F” rendszer nagyon jól számítható a hozzá tartozó többállapotú „*k* az *n*-ből: G” rendszeren keresztül.

5.4.3 Többállapotú súlyozott „*k* az *n*-ből” rendszermodell

Tegyük fel, hogy *n* darab komponens van a rendszerben, amelyek mindegyik M + 1 állapotban lehet (0, 1, …, M). Amikor az *i* komponens a *j* állapotban van, akkor hozzá egy w_{ij} súly tartozik.

A bináris súlyozott „*k* az *n*-ből: G” rendszerek esetében a rendszer összsúlya a működő komponenseihez tartozó súlyok összege, amikor egy komponens meghibásodik, akkor a hozzá tartozó súly 0 lesz.

Többállapotú környezetben egy másképp alakul, mivel a komponens többféle állapotban lehet. Amikor különböző állapotokban van, különböző mértékben járul hozzá a rendszer teljesítményéhez, így különböző súlyokkal rendelkezik. Amikor teljesen működésképtelenné válik, akkor a hozzá tartozó súly ebben az esetben is 0 lesz.

Két fő modellt dolgoztak ki a többállapotú súlyozott „*k* az *n*-ből: G” rendszerekre. Az első modell szerint:

A rendszer a *j* vagy afölötti állapotban van, ha a komponensek összsúlya nagyobb vagy egyenlő, mint egy előre meghatározott k_j érték.

Legyen ϕ a rendszer állapotát leíró függvény, W pedig a komponensek összsúlya. Ekkor a fenti definíció alapján $P\{\phi \geq j\} = P\{W \geq k_j\}$. Mivel a rendszer lehető legrosszabb állapota a 0, így $P\{\phi \geq 0\} = 1$.

2000 – ben mutatták be az általános többállapotú „*k* az *n*-ből” rendszer modelljét, amelyben ahhoz, hogy a rendszer ne legyen egy megadott *j* állapottól alacsonyabb szinten, minimum k_j olyan komponenssel kell rendelkeznie, amelyek nincsenek a *j* – től lejjebb lévő állapotban. Eszerint a szemlélet szerint azok a komponensek, amelyek a *j* alatti állapotok valamelyikében vannak, nem járulnak hozzá ahhoz, hogy a rendszer minimum a *j* állapotban legyen.

Ezen alapszik a többállapotú súlyozott „*k* az *n*-ből: G” rendszerek másik modellje. Ebben a modellben ahhoz, hogy a rendszer egy *j* vagy afölötti állapotban legyen, a *j*, vagy afölötti állapotban lévő komponensei összsúlyának el kell érnie egy előre meghatározott k_j értéket. A két modell között abban van a különbség, hogy a j állapottól alacsonyabb állapotban lévő komponensek hozzájárulnak – e a rendszer állapotához.

Ezen rendszer formális definíciója:

A rendszer a *j*, vagy afölötti állapotban van, ha azon komponensei összsúlya, amelyek szintén a *j*, vagy afölötti állapotban vannak, minimum egy előre meghatározott k_j érték.

Jelölje ϕ a rendszer állapotát leíró függvényt, W_j pedig azon komponensek összsúlyát, amelyek legalább a *j* állapotban vannak. Ekkor $P\{\phi \geq j\} = P\{W_j \geq k_j\}$.

A többállapotú súlyozott „*k* az *n*-ből" rendszereket alkalmaznak a repülőgépeken, telekommunikációs hálózatokban, közlekedési rendszerekben, űrhajókban és informatikai rendszerekben is.

Egy hétköznapi példa az első modellel leírt többállapotú súlyozott „*k* az *n*-ből" rendszerekre a repülőgép. Egy utasszállító repülőgép általában több motorral rendelkezik. Ezeket tekinthetjük komponenseknek, a repülőgépet pedig a rendszernek. Minden motor képes teljes kapacitással működni, részlegesen működni, vagy le is állhat, a különböző állapotokban különböző nagyságú tolóerő kifejtésére képes. A repülőgép teljes tolóerejét a motorok tolóerejének összege adja. Nem számít, hogy egy motor teljes kapacitással működik, vagy éppen leállt, a teljes tolóerő a motorok által kifejtett tolóerők összege lesz.

5.5 Megbízhatóság számítása a többállapotú „*k* az *n*-ből" rendszerekben

A többállapotú „*k* az *n*-ből" rendszerek esetében a megbízhatóság kiszámítása rekurzív függvények segítségével történik. Az ilyen rendszereknek több állapota van. A megbízhatóság meghatározása tehát azt jelenti, hogy meg kell határozni annak a valószínűségét, hogy a rendszer az egyes állapotok valamelyikében van. Az $r_{s,j}$ jelöli annak a valószínűségét, hogy a rendszer a *j* állapotban van. Így meg kell határoznunk az $r_{s,j}$ - t minden *j* – re.

Néhány esetben nem az a fontos, hogy a rendszer milyen valószínűséggel van egy megadott állapotban, hanem, hogy milyen valószínűséggel van egy megadott állapot fölött, vagy alatt. Ezeket a $P_{s,j}$ és a $Q_{s,j}$ jelöli.

5.5.1 A rekurzív algoritmusok alapvető elemei

A rekurzív algoritmusok használata nagyon hatékony módja a szoftvermegbízhatóság számításának. A többállapotú „*k* az *n*-ből” rendszerek esetében a megbízhatóságot hatékonyan számító algoritmusok mindegyik rekurzív függvényeket használ. A rekurzív algoritmusoknak három fő eleme van:

1, Rekurzív függvény

2, Frissítő algoritmus

3, Peremfeltételek

5.5.1.1 A rekurzív függvény

A rekurzív függvény az a központi függvény a rekurzív algoritmusban, amely folyamatosan önmagát hívja meg. Több paraméterrel is rendelkezik, amelyek a rekurzív algoritmus során változhatnak. A bináris „*k* az *n*-ből” rendszerek esetében használt algoritmusokban a rekurzív függvény az R(n, k), amelyben *n* és *k* a paraméterek.

5.5.1.2 A frissítő algoritmus

A frissítő algoritmus dönti el, hogy hogyan hívja meg önmagát a rekurzív függvény. Vagyis a frissítő algoritmus mondja meg, hogy milyen kapcsolat van a meghatározott paraméterekkel megadott rekurzív függvény és az eltérő paraméterekkel rendelkező között, amely általában kisebb komplexitású.

A A bináris „*k* az *n*-ből” rendszerekben az R(n, k) két másik rekurzív függvény segítségével: R(n – 1, k) és R(n – 1, k – 1) számítható, amelyek paramétere kisebb, így könnyebben lehet őket számítani.

5.5.1.3 Peremfeltételek

Amikor valamelyik peremfeltétel teljesül, akkor a rekurzív függvény értéke egy előre maghatározott érték, vagy egy speciális egyszerű módon kiszámítható.

A bináris „*k* az *n*-ből” rendszerek esetében is léteznek ilyen peremfeltételek, méghozzá kettő. Az első, hogy amennyiben a k = 0, akkor a rekurzív függvény értéke 1. A másik peremfeltétel pedig, hogy ha a 0 < n < k, akkor az R(n, k) értéke 0.

5.5.2 Megbízhatóság számítása Huang modelljében

Minden a Huang által definiált többállapotú „*k* az *n*-ből: G” rendszerhez létezik egy vele ekvivalens többállapotú „*k* az *n*-ből: F” rendszer. A többállapotú „*k* az *n*-ből: G” rendszer megbízhatóságának számítása elvégezhető a többállapotú „*k* az *n*-ből: F” rendszeren keresztül.

A nominális növekedő többállapotú „*k* az *n*-ből: F” rendszer hasonló a növekedő többállapotú „*k* az *n*-ből: F” rendszerhez, annyi eltéréssel, hogy a valószínűsége annak, hogy egy komponens egy adott állapotban van kisebb, mint 1.

A megbízhatóság számításához meg kell határozni a rendszer minden *j* állapotára a $Q_{s,j}$ valószínűséget, annak a valószínűségét, hogy a rendszer a *j* alatti állapotban van.

A rekurzív függvénynek négy paramétere van, a függvényt: Q(m, N, **k**, **p**) - vel jelöljük. Az m paraméter a nominális növekedő többállapotú „*k* az *n*-ből: F” rendszer összes lehetséges állapota mínusz 1, az N a rendszer komponenseinek száma, a ***k*** a nominális növekedő többállapotú „*k* az *n*-ből: F” rendszer **k** vektora, ahol $k = (k_1, k_2, \ldots, k_m)$, valamint ***p*** a rendszer valószínűségi vektora, amelyre $p = (p_0, p_1, \ldots, p_m)$.

Ezt a rekurzív függvény arra tervezték, hogy megadja annak a valószínűségét, hogy a rendszer a 0 állapotban van, vagyis nem működik. A $Q_{s,j}$ értéket úgy tudjuk kiszámítani, hogy transzformáljuk a rendszert egy nominális növekedő többállapotú „*k* az *n*-ből: F” rendszerré, a *j* állapotra nézve, ezután pedig elvégezzük a számításokat a rekurzív függvény alapján. Ez a rekurzív algoritmus megköveteli, hogy a komponensek egymástól függetlenek, és azonos eloszlással rendelkezzenek.

A transzformáció során a lehetséges állapotok száma (m) (M + 1) – j – vel lesz egyenlő, a k vektornak pontosan ennyi eleme lesz, amelyek a *j*, vagy afölötti állapotokhoz tartozó k_i értékek. A p vektorban szerepelnek a j, vagy afölötti állapotokhoz tartozó p_i értékek, az az alattiakat pedig összegezve tartalmazza a p_0 nominális 0 állapot.

A frissítő algoritmus a következő:

$$Q(m, N, k, p) = \sum_{i=k_1}^{k_2-1} \binom{N}{i} p_0^i * Q(m-1, N-i, \dot{k}, \dot{p}) + Q(m-1, N, \ddot{k}, \ddot{p}),$$

ahol $\dot{k} = (k_2 - i, k_3 - i, \ldots, k_m - i)$, $\dot{p} = (p_1, p_2, \ldots, p_m)$, $\ddot{k} = (k_2, k_3, \ldots, k_m)$, $\ddot{p} = (p_0, p_1 + p_2, p_3, \ldots, p_m)$.

A peremfeltétele a rekurzív algoritmusnak: $Q(1, N, k, p) = \sum_{i=k_1}^{N} \binom{N}{i} p_0^i * p_1^{N-i}$. Amint láthatjuk ez az algoritmus az m paraméteren rekurzív, tehát nagyon jól használható sok komponensből álló nagy rendszerek vizsgálatakor is anélkül, hogy a számítási idő szignifikáns mértékben növekedne.

A $Q_{s,j}$ kiszámítását egy példán keresztül mutatom be:

Legyen adott egy növekedő többállapotú „*k* az *n*-ből: F" rendszer 10 egymástól független és azonos eloszlással rendelkező komponenssel, és 4 lehetséges állapottal. Ekkor n = 10, M = 3. A **k** vektor legyen k = (3, 6, 8), a **p** vektor pedig p = (0.1, 0.3, 0.4, 0.2).

Ahhoz, hogy ki tudjuk számolni a $Q_{s,3} - t$ át kell térnünk egy nominális növekedő többállapotú „*k* az *n*-ből: F" rendszerre, a 3 – as állapotra vonatkozóan. Ezt a következőképpen tehetjük meg: m = 1, N = 10, k = (k_3) = (8) p = ($p_0 + p_1 + p_2$, p_3) = (0.8, 0.2). A rekurzív algoritmust használva $Q_{s,3} = Q(m, N, k, p)$, amely érték 0.6678.

A $Q_{s,2}$ kiszámításához szintén át kell térni egy másik rendszerre, de immár a 2 – es állapotra nézve. Ekkor m = 2, N = 10, k = (k_2, k_3) = (6, 8), p = ($p_0 + p_1$, p_2, p_3) = (0.4, 0.4, 0.2). Szintén a rekurzív algoritmust használva, 0.1523 – t kapunk eredményül.

A $Q_{s,1}$ számításakor szintén át kell térnünk egy másik rendszerre méghozzá az 1 – es állapotra vonatkozóan. Ebben az esetben m = 3, N = 10, k = (k_1, k_2, k_3) = (3, 6, 8), p = (p_0, p_1, p_2, p_3) = (0.1, 0.3, 0.4, 0.2). Ekkor az eredmény 0.0308.

Abban az esetben, ha a komponensek csak egymástól függetlenek, egy másik rekurzív algoritmust kell használnunk. Ekkor a rekurzív függvény: Q(n, **k**, **P**), amelyben

n: a rendszerben található komponensek száma

k: a rendszer **k** vektora, $k = (k_1, k_2, \ldots, k_M)$, ahol M a lehetséges állapotok száma

P: a komponens állapotok eloszlási mátrixa a rendszerre vonatkozóan

$$P = \begin{pmatrix} p_{1,0} & p_{1,1} & \cdots & p_{1,M} \\ p_{2,0} & p_{2,1} & \cdots & p_{2,M} \\ \cdots & \cdots & \cdots & \cdots \\ p_{n,0} & p_{n,1} & \cdots & p_{n,M} \end{pmatrix}.$$

A rekurzív függvényt szintén a 0 állapot valószínűségét határozza meg. A $Q_{s,j}$ kiszámításakor ebben a módszerben is át kell térnünk egy olyan rendszerre, amelyben a *j* alatti állapotokat egy 0 – s nominális állapottá kombináljuk.

A frissítő algoritmus a következő:

$$Q(n,k,P) = \sum_{j=0}^{M} p_{n,j} * Q(n-1,k^j,P^j).$$

Az alapötlet ebben az esetben is az, mint az előzőben, hogy meghatározzuk az n komponensre, hogy mely állapotokban lehet, és ezek segítségével számítjuk ki a rendszer megbízhatóságát oly módon, hogy visszavezetjük egy n – 1 komponenssel rendelkező rendszer megbízhatóságára.

Minden egyes j – re át kell alakítanunk a jobb oldalon szereplő k^j és P^j – ket a következőképpen:

Ha $j \neq M$

$$k_l^j = k_l - 1, \; l \geq j + 1 - \text{re},$$

$$k_l^j = k_l, \; l < j + 1 - \text{re}.$$

Azért így kell számítani, mert ha az n komponens a j szinten van, akkor már eggyel kevesebb komponens szükséges a j fölötti szinteken. Ha j = M, $k_l^j = k_l, \; 1 \leq l \leq M - \text{re}$. A P^j – t úgy kapjuk, hogy kitöröljük az n. sort a P mátrixból, így a P^j egy n – 1 x M + 1 – es mátrix lesz.

Létezik egy speciális eset, amikor a k^j és P^j előállításakor, amelyben egy bizonyos állapotot „elnyelni” a közvetlenül alatta lévő állapot. Ekkor további transzformációkat kell elvégezni a k^j és P^j – n.

Ez a speciális eset az, amikor $k_h^j = k_{h+1}^j$ egy $0 \leq h \leq M - 1$ állapotra. Ebben az esetben a h+1 – es állapotot elnyeli a h állapot. Ekkor a $k_{h+1}^j - t$ ki kell törölni a a k^j – ből, $p_{i,h}^j = p_{i,h}^j + p_{i,h+1}^j, \; 1 \leq i \leq n-1,$ valamint a h+1. oszlopot törölni kell a P^j – ből. Ekkor

a lehetséges állapotok száma M + 1 – ről M – re csökken. Ez egyébként a fő oka annak, hogy a számítási idő nem nő exponenciálisan az n növekedésével.

Ha így állítjuk elő k^j és P^j – t, akkor k^j mindig szigorúan növekvő vektor lesz.

Két peremfeltétel van ebben az esetben is, amelyek a következők:

1, Ha $k_M > n$, akkor Q(n, **k**, **P**) = 0.

2, Ha M = 1, akkor a növekedő többállapotú „*k* az *n*-ből: F” rendszer lényegében egy bináris „*k* az *n*-ből: F” rendszer, amely független komponensekkel rendelkezik. Így az eddig használt algoritmusok használhatók a bináris rendszerekben is.

A következő példában tekintsünk egy növekedő többállapotú „*k* az *n*-ből: F” rendszert, független komponensekkel. Legyen 5 független komponense, és 5 lehetséges állapota. Ekkor legyen a **k** = (k_1, k_2, k_3, k_4) = (1, 2, 3, 4). Az állapot eloszlási mátrix:

$$P = \begin{pmatrix} 0.1 & 0.2 & 0.1 & 0.4 & 0.2 \\ 0.2 & 0.1 & 0.1 & 0.3 & 0.3 \\ 0.1 & 0.2 & 0.1 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0.1 & 0.1 & 0.2 \\ 0.1 & 0.1 & 0.2 & 0.2 & 0.4 \end{pmatrix}.$$

A rekurzív formulát használva a következő eredményre juthatunk:

$\mathbf{Q_s}$ = (0.2526, 0.2939, 0.3350, 0.5261) .

$\mathbf{r_s}$ = (0.2526, 0.0413, 0.0412, 0.1910, 0.4739) .

5.5.2 Megbízhatóság számítása Tian modelljében

Hasonlóképpen a nominális növekedő többállapotú „*k* az *n*-ből: F” modellhez definiáljuk a nominális csökkenő többállapotú „*k* az *n*-ből: G” modellt a következőképpen:

A nominális csökkenő többállapotú „*k* az *n*-ből: G” rendszer hasonlító a csökkenő többállapotú „*k* az *n*-ből: G” rendszerhez, annyi eltéréssel, hogy annak a valószínűsége, hogy egy komponens egy adott állapotban van kisebb, mint 1.

A nominális csökkenő többállapotú „*k* az *n*-ből: G” rendszer annak az eredménye, hogy a komponensek lehetséges állapotai közül csak néhánnyal foglalkozunk, vagy összevonunk több szomszédosat. A többállapotú „*k* az *n*-ből: G” rendszerek számításában fontos szerepet játszik a nominális csökkenő többállapotú „*k* az *n*-ből: G” rendszer.

Egy általános többállapotú „*k* az *n*-ből: G” rendszerhez tartozó $P_{s,j} - t$ úgy számíthatjuk ki bármely j állapotra, hogy áttérünk egy nominális csökkenő többállapotú „*k* az *n*-ből: G” rendszerre, és ezt a rendszert vizsgáljuk a legmagasabb nominális állapotban.

Ahhoz, hogy az állapotok valószínűségének eloszlását megkapjuk, ki kell számítani a $P_{s,1}, P_{s,2}, \ldots, P_{s,M} - et$, ahol $P_{s,j}$ annak a valószínűsége, hogy a rendszer a *j*, vagy afölötti állapotban van.

Amennyiben a komponensek egymástól függetlenek és azonos eloszlással rendelkeznek, a rendszer jellemzéséhez használt rekurzív függvény: P(n, **k**, **P**), ahol

n: a rendszerben lévő komponensek száma

k: a rendszer **k** vektora, $k = (k_1, k_2, \ldots, k_M)$, ahol M a lehetséges állapotok száma mínusz 1.

3, a komponens állapotok eloszlási mátrixa a rendszerre vonatkozóan

$$P = \begin{pmatrix} p_{1,0} & p_{1,1} & \cdots & p_{1,M} \\ p_{2,0} & p_{2,1} & \cdots & p_{2,M} \\ \cdots & \cdots & \cdots & \cdots \\ p_{n,0} & p_{n,1} & \cdots & p_{n,M} \end{pmatrix}.$$

A P(n, **k**, **P**) a legnagyobb nominális állapothoz tartozó valószínűséget adja meg a rendszerre vonatkozóan.

A frissítő algoritmus hasonló a Huang modelljében használttal:

$$P(n,k,P) = \sum_{j=0}^{M} p_{n,j} * P(n-1, k^j, P^j).$$

Annyi eltéréssel, hogy ebben az esetben a k^j és P^j generálásakor a következő módon járunk el:

Ha $0 < j < M$,

$$k_l^j = k_l, \quad l > j - \text{re},$$

$$k_l^j = k_l - 1, \quad l \le j - \text{re}.$$

Annak alapján hogy amennyiben az n komponens a j állapotban van, úgy a j alatti állapotokban szükséges komponensek számát csökkentenünk kell 1 – gyel. Ha j = M, akkor $k_l^j = k_l - 1, \ 1 \le l \le M$. Ha j = 0, akkor $k_l^j = k_l, \ 1 \le l \le M$. A P^j – t ebben az esetben is úgy kapjuk, hogy kitöröljük az n. sort a P mátrixból, így a P^j egy n – 1 x M + 1 – es mátrix lesz.

Ekkor is létezik egy speciális eset, amikor a h állapotot elnyeli egy szomszédos állapotan, azonban ebben az esetben ez a h – 1 állapot. Így a k^j vektorból a k^j_{h-1} elemet kell törölni, a P^j – ből pedig a h – 1. oszlopot. Ezáltal a lehetséges állapotok száma szintén csökken eggyel, így M darab lesz.

Ha így állítjuk elő k^j és P^j – t, akkor k^j itt is mindig szigorúan növekvő vektor lesz.

A peremfeltételek a következők:

1, Ha $k_1 > n$, akkor P(n, **k**, **P**) = 0.

2, Ha M = 1, akkor a növekedő többállapotú „*k* az *n*-ből: G” rendszer lényegében egy bináris „*k* az *n*-ből: G” rendszer, amely független komponensekkel rendelkezik. Így az eddig használt algoritmusok használhatók az ilyen bináris rendszerekben is.

Tekintsünk egy csökkenő többállapotú „*k* az *n*-ből: G” rendszert, független komponensekkel. Legyen 5 független komponense, és 5 lehetséges állapota. Ekkor legyen a **k** = (k_1, k_2, k_3, k_4) = (4, 3, 2, 1). Az állapot eloszlási mátrix:

$$P = \begin{pmatrix} 0.2 & 0.4 & 0.1 & 0.2 & 0.1 \\ 0.3 & 0.3 & 0.1 & 0.1 & 0.2 \\ 0.4 & 0.2 & 0.1 & 0.2 & 0.1 \\ 0.2 & 0.1 & 0.1 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0.2 & 0.1 & 0.1 \end{pmatrix}.$$

A rekurzív formulát használva a következő eredményre juthatunk:

$\mathbf{Q_s}$ = (0.5261, 0.3350, 0.2939, 0.2526) .

$\mathbf{r_s}$ = (0.4739, 0.1910, 0.0412, 0.0413, 0.2526) .

5.6 Konklúzió

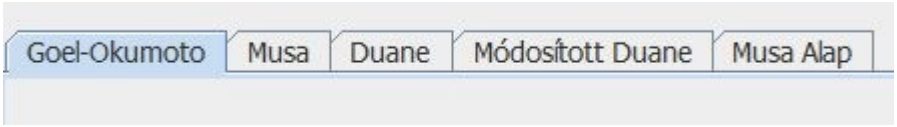
A „*k* az *n*-ből” struktúra nagyon népszerű redundancia struktúra, és széles körben használatos nagyon sok rendszerben. Nagyon sok gyakorlati komponens és rendszer képes különböző állapotokban működni. Ezek az állapotok jelenthetik a teljesítményt, amely leadására az adott pillanatban képesek. Több rekurzív eljárást is kifejlesztettek tipikusan az ilyen rendszerek megbízhatóságának számítására, így az ilyen szoftverek megbízhatóság már nagyon hatékonyan számítható.

6.0 A megbízhatóságot számító analitikus program

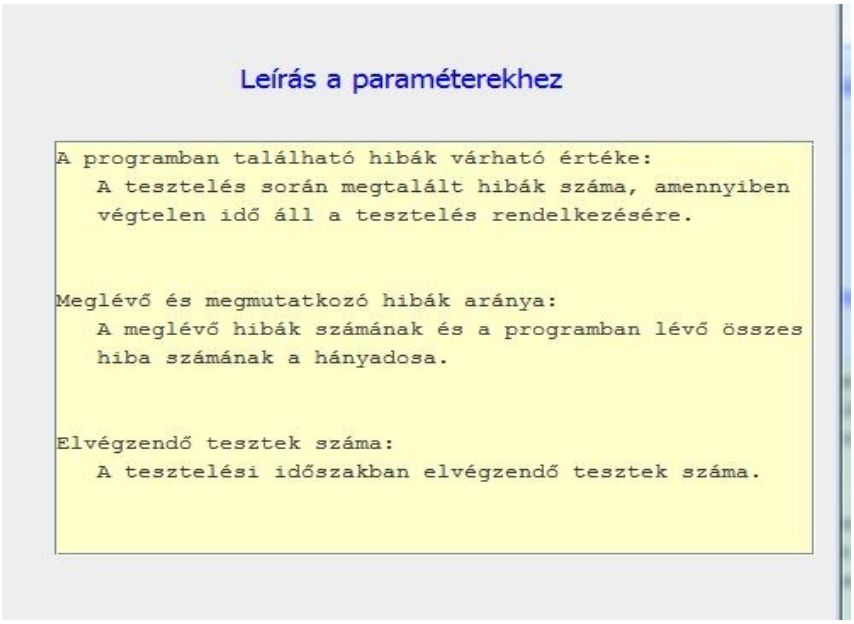
6.1 A program működése

A dokumentumhoz tartozik egy program, amely a Goel – Okumoto, Musa, Duane, Módosított Duane, valamint az Alap Musa modellben való számításokban segít. Segítségével kiszámíthatjuk az egyes modellekben, azoktól függően a meghibásodási intenzitást, illetve a programban megmaradó hibák számát.

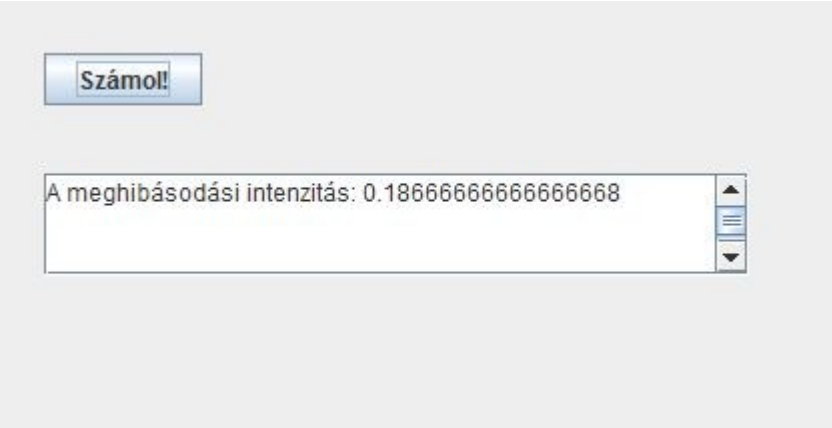
A program indítása után fülek segítségével választhatjuk ki, hogy melyik modellben szeretnénk számításokat végezni. A megfelelő fül kiválasztása után a megjelenő lapon kell megadni a rendszerre jellemző paramétereket.



A paraméterek megértéséhez, kiválasztásához az egyes fülek tartalmaznak egy, a paraméterekre vonatkozó leírást.



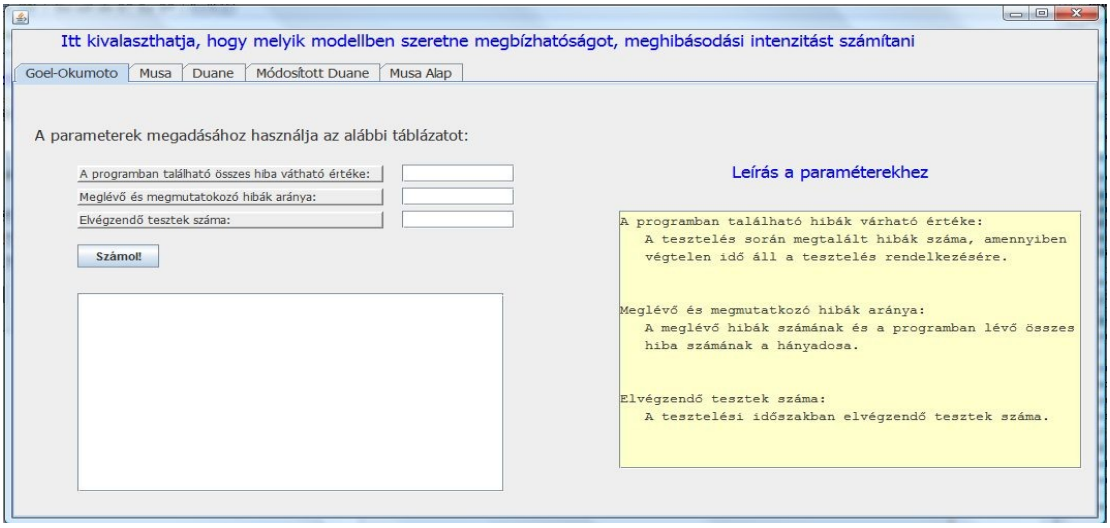
A szükséges adatok megadása után a számol gomb megnyomásával a program a bemenő adatok alapján elvégzi a számításokat, amelyek eredményét az alsó részben prezentálja szöveges formában.



Amennyiben másik modellben szeretnénk számításokat végezni, vagy ugyanabban a modellben szeretnénk ismét, a friss adatok beírásával, és a számol gomb ismételt megnyomásával, valamit másik fülre való navigálás, és azon a szükséges paraméterek megadása után, megtehetjük.

6.2 A program jellemzői

6.2.1 Goel – Okumoto modell



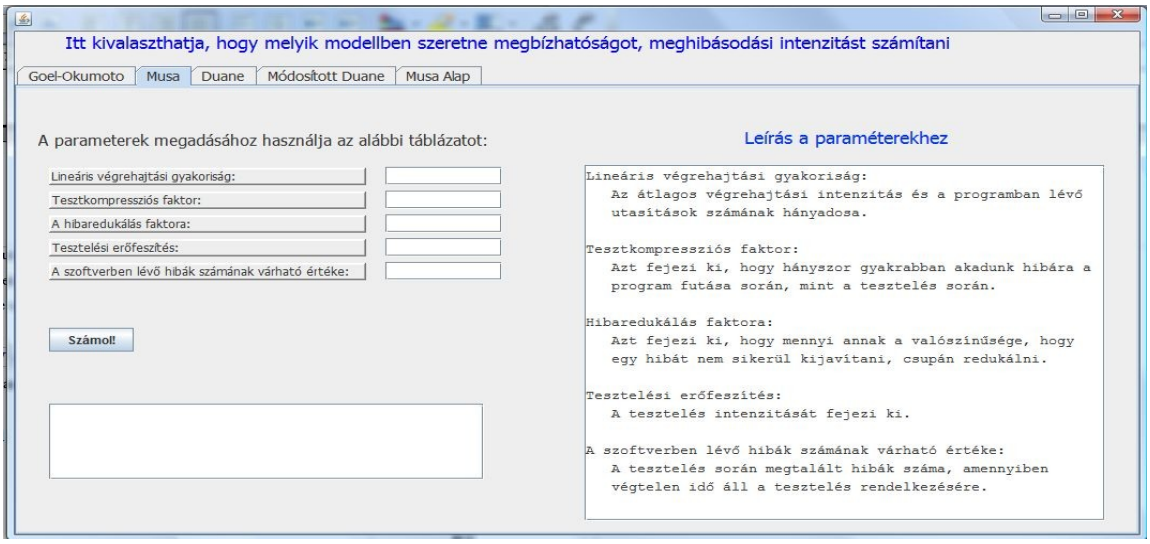
A szükséges paraméterek megadása után – amelyekhez a fülön találunk egy segédletet – a szoftver kiszámítja az adott paraméterekkel vonatkozó szoftverben várhatóan megmaradó hibák számát, az egyes tesztek elvégzése után.

Ezt a számítást a

$$E\,\bar{N}(t) = E\left[N(\infty) - N(t)\right] = m(\infty) - m(t) = a - a(1 - e^{-bt}) = ae^{-bt}$$

képlet alapján végzi, a megadott paraméterek felhasználásával.

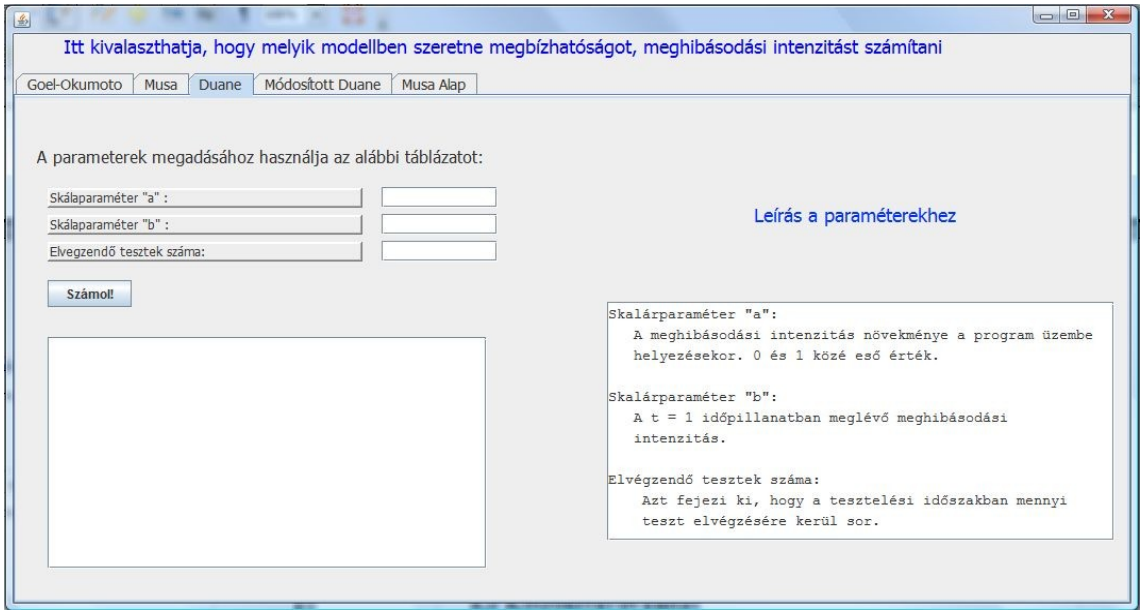
6.2.2 Musa modellje



A szükséges paraméterek megadása után – amelyekhez a fülön találunk egy segédletet – a szoftver kiszámítja az adott paraméterekkel vonatkozó szoftverben várhatóan megmaradó hibák számát.

Ezt a $\lambda(\iota) = fK\left[N_0 - \mu(\iota)\right]$, képlet alapján végzi. Ebben a képletben az f, K N_0 paramétereket a felhasználó adja meg, a $\mu(\iota)$-t pedig a szoftver számítja az $N_0 * \left(1 - e^{-BCFK}\right)$ képlet alapján.

6.2.3 Duane modellje



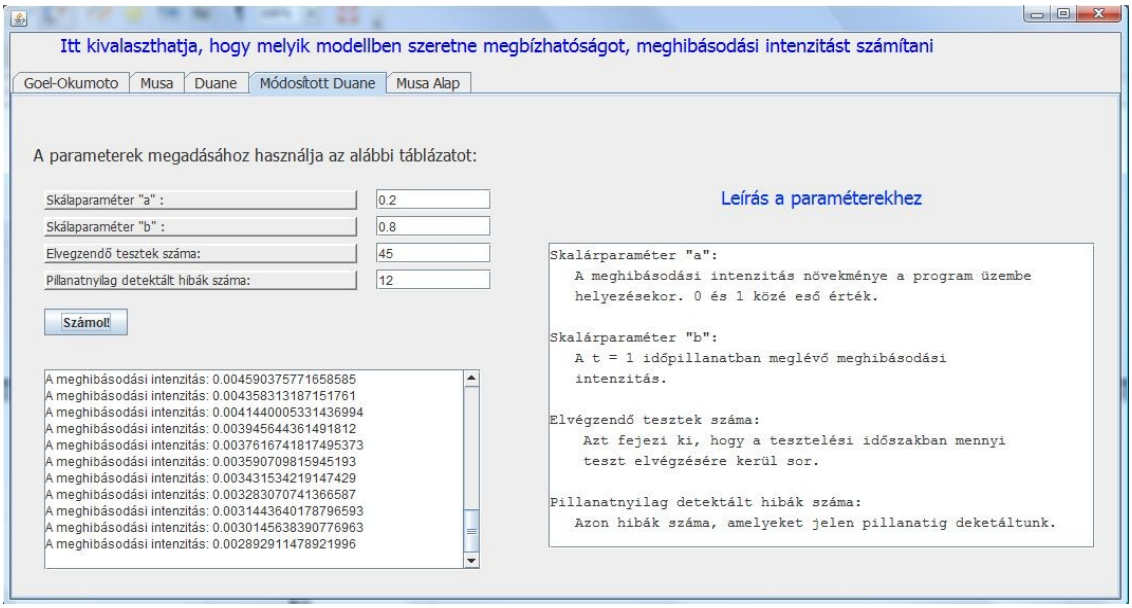
Ebben a modellben a két skalárparaméter, valamint az elvégzendő tesztek számának megadása után, a program kiszámítja az egyes tesztesetek után várható meghibásodási intenzitását a vizsgált szoftvernek.

A hibák megjelenésének intenzitása a t időpillanatban:

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1}, \quad \alpha > 0, \beta > 0.$$

A meghibásodási intenzitás ezen képlet alapján kerül kiszámításra.

6.2.4 Módosított Duane modell



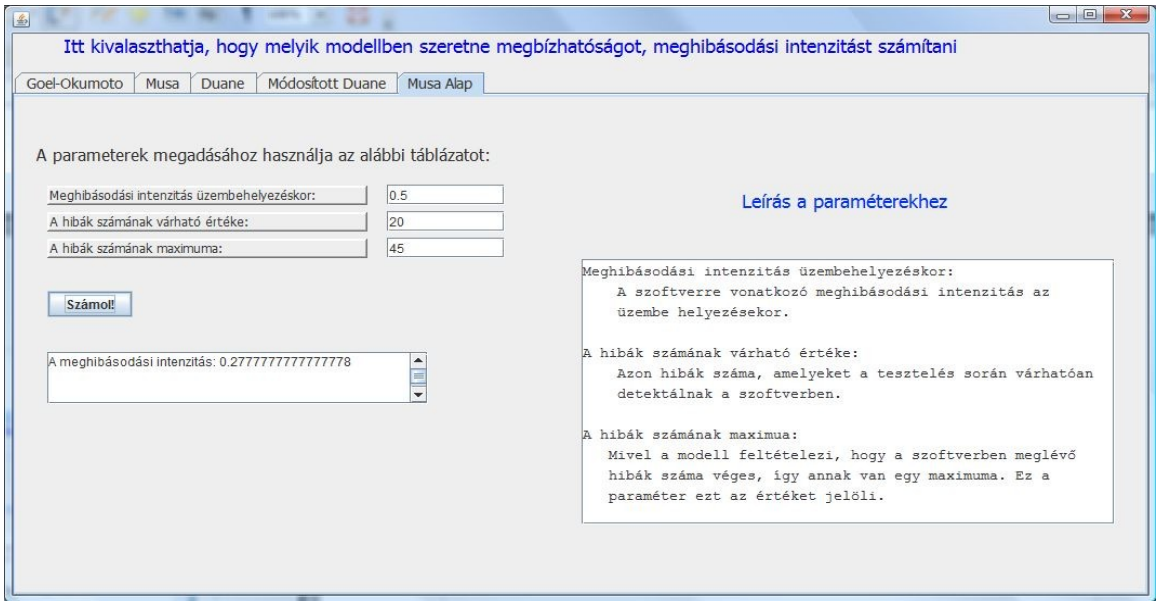
A módosított Duane modellben a módosítást egy új paraméter bevezetése jelenti, amely szerepet vállal a tesztek után kiszámított intenzitás meghatározásakor. Ennek a modellnek előnye a Duane modellel szemben, hogy a kezdeti időpillanatban a meghibásodási intenzitás egy véges érték, ellentétben a Duane modellel, ahol ez végtelen.

Ebben az esetben a hibák megjelenésének intenzitása:

$$\lambda(t) = m'(t) = k\,\beta\,\alpha^{\beta}(\alpha + t)^{-1-\beta}.$$

Ekkor a 0. időpillanatban a meghibásodási intenzitás $\frac{k\,\beta}{\alpha}$, ami egy véges érték, és az intenzitás tart 0 – hoz, ahogyan a t – vel tartunk a végtelenbe.

6.2.5 Alap Musa modell



Ebben a modellben a meghibásodási intenzitást számíthatjuk a kezdeti meghibásodási intenzitás, valamint a hibák várható értékének, és maximumának megadásával.

Ebben az esetben a meghibásodási intenzitást a következőképpen kapjuk:

$$\lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{\nu_0}\right), \quad \text{ahol}$$

$\lambda(\mu)$ a meghibásodási intenzitás

λ_0 meghibásodási intenzitás a szoftver üzembehelyezésekor

μ a várható hibák száma egy megadott időintervallumban

ν_0 az összes megjelenő hibának a száma, ha az idővel tartunk a végtelenbe.

7.0 Konklúzió

A dolgozat célja, hogy átfogó képet adjon a szoftvermegbízhatóságról, illetve különböző modelleket mutasson be, amelyeket a mai gyakorlatban a legtöbbször használnak.

A dolgozatban lévő számítások segítséget adhatnak a tervezett szoftver megbízhatóságának számításában, illetve a különböző modellek segítségével más - más szemszögből, különböző paraméterek használatával számítható megbízhatóság, így arról átfogóbb képet kaphat a felhasználó.

A mellékelt programban lévő számítások hasznosak lehetnek a tesztelés, illetve a későbbi támogatás tervezésekor.

A dolgozat célja; a szoftvermegbízhatóság bevezetése, valamit annak számításához különböző manapság használatos modellek megadása, sikeres volt. A mellékelt program pedig hasznos lehet nemcsak a megbízhatóság számításakor, hanem a különböző szemszögből történő vizsgálatakor is.

Így összességében úgy érzem, sikeres lett a célkitűzés megvalósítása.

8.0 Köszönetnyilvánítás

Ezúton szeretném megköszönni dr. Sztrik János tanárúrnak, a témavezetőmnek, hogy segített a dolgozat elkészítésében, és a segítőkész hozzáállását az egész diplomamunkával kapcsolatban.

Valamint a családomnak a támogatást, és a megértő hozzáállást az egyetemen töltött éveim alatt. Külön köszönöm menyasszonyomnak: Sebestyén Gabriellának, a lelki és szakmai támogatást, amellyel hozzásegített eredményeim eléréséhez.

9.0 Irodalomjegyzék

[1] Hoang Pham:
Recent Advances in Reliability and Quality in Design
Kiadó: Springer
London, 2008

[2] dr. Kun István, dr. Szász Gábor, dr. Zsigmond Gyula:
Minőség és Megbízhatóság I
Kiadó: LSI Informatikai Okaktóközpont
Budapest, 2004

[3] Douglas N. Arnold:
Two disasters caused by computer arithmetic errors
Letöltés helye: http://www.ima.umn.edu/~arnold/455.f96/disasters.html

[4] Jiantao Pan:
Software Reliability
Letöltés helye: http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

[5] Alan Wood:
Software Reliability Growth Models
Letöltés helye: http://www.hpl.hp.com/techreports/tandem/TR-96.1.pdf

[6] M. Xie:
Software Reliability Modelling
Kiadó: World Scientific Publishing Co. Pte. Ltd.
Singapore, 1991

[7] Kun István:
Informatikai rendszerek megbízhatóságának matematikai modellezése
Letöltés helye: http://robothadviseles.hu/pres/Kun_Istvan.pdf

10.0 Függelék

Nem – homogén Poisson-folyamat:

A nem-homogén Poisson-folyamat olyan Poisson-folyamat, amely paramétere nem konstans, hanem egy függvény. Az erre épülő modellek igen fontosak és széles körben elterjedtek a szoftvermegbízhatóságban.

Weibull folyamat:

A Weibull-eloszlás nagy fontossággal bír a megbízhatóságelméletben. Ezt alapvetően az eloszlás két fontos tulajdonsága magyarázza. A Weibull-eloszlás alkalmas mind csökkenő, mind a növekvő, mind az állandó meghibásodási rátájú folyamatok leírására, másrészt bizonyos paraméterezés esetén közelit a normális illetve exponenciális eloszláshoz.

A Weibull folyamat olyan nem homogén Poisson-folyamat, amely intenzitásfüggvénye:

$$\lambda(t) = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1}.$$