OKOSVÁROS-TECHNOLÓGIÁK XX.

Kvalitatív módszerek használata hibaterjedési vizsgálatokban

Gönczy László – Jakab László – Kocsis Imre

Nemzeti Közszolgálati Egyetem
Budapest

SZÉCHENYI 2020
Magyarország Kormánya
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

OKOSVÁROS-TECHNOLÓGIÁK
A technológia fejlődésének irányai és hatása
XX. kötet

Sorozatszerkesztő:

Sallai Gyula

Gönczy László – Jakab László – Kocsis Imre

KVALITATÍV MÓDSZEREK HASZNÁLATA HIBATERJEDÉSI VIZSGÁLATOKBAN

Nemzeti Közszolgálati Egyetem
Közigazgatási Továbbképzési Intézet
Budapest, 2020

A kötet a Nemzeti Közszolgálati Egyetem **KÖFOP-2.1.2-VEKOP-15-2016-00001**
„**A jó kormányzást megalapozó közszolgálat-fejlesztés**” projektje keretében,
a Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Karán létesült „**Okos város – okos közigazgatás**”
kutatóműhelyben (2017/162/BME-VIK) készült.

Szakmai lektor:
Pataricza András egyetemi tanár, BME-VIK

Szerkesztette:
Gönczy László egyetemi adjunktus, BME-VIK

A kézirat lezárásának dátuma:
2018. március 31.

© Nemzeti Közszolgálati Egyetem
Közigazgatási Továbbképzési Intézet, 2020
© Gönczy László, 2020
© Jakab László, 2020
© Kocsis Imre, 2020

A mű szerzői jogilag védett. Minden jog, így különösen a sokszorosítás, terjesztés és fordítás joga fenntartva. A mű a kiadó írásbeli hozzájárulása nélkül részeiben sem reprodukálható, elektronikus rendszerek felhasználásával nem dolgozható fel, azokban nem tárolható, azokkal nem sokszorosítható és nem terjeszthető.

TARTALOMJEGYZÉK

Bevezetés **6**

1. A hibaterjedési elemzés jelentősége, szabványos folyamatai **7**

2. A hibaterjedés-elemzés alapfogalmai **23**

2.1. Hibaok, hibás állapot, hibahatás 23

2.2. Rendszerszintű hibaterjedés 24

2.3. Rendszerszintű hibaterjedés korlátozása 25

3. Modern hibaterjedés-elemzés **27**

3.1. Kvalitatív hibaautomaták 28

3.2. Statikus szindróma alapú hibaterjesztés 28

3.3. EPA az okos város módszertanokban 31

4. Hibaterjedés elemzési módszerek alkalmazása okos város környezetben ... **33**

4.1. Sorbakapcsolt (kaszkád) hibahatások 33

4.2. Kritikus infrastruktúrák folyamatainak ellenőrzése szimulációval 34

4.3. Kvalitatív hibaterjedési vizsgálatok támogatása mérnöki modellekkel 35

4.4. Modellezési nyelvek 36

4.4.1. ystems Modelling Language (SYSML) *36*

4.4.2. Architecture Analysis & Design Language (AADL) *37*

4.4.3. Üzleti folyamatok webszolgáltatás alapon: BPEL *38*

4.4.4. BPMN *39*

5. Hibaterjedési vizsgálat „intelligens kereszteződés” szolgáltatásaiban (esettanulmány) **40**

5.1. A Tilos jelzésen áthaladás (Red Light Running) alkalmazás hibamodellje és elemzése 43

5.2. A kritikus besorolás hatása a rendszer fejlesztésére 46

5.3. Az alkalmazás/szolgáltatás és az infrastruktúra kritikusságának egymásra hatása 50

Irodalomjegyzék **51**

BEVEZETÉS

Jelen monográfia célja a kvalitatív modellezési és elemzési módszerek áttekintése, elsősorban az okos várost támogató kritikus infrastruktúrák és a ráépülő alkalmazások szempontjából. A monográfia áttekinti a hibaterjedési elemzés módszereit és jelentőségét, emellett bemutatja a hibaterjedés-elemzésnek azt a módszerét, amellyel hatékonyan lehet heterogén, egymással összekötött infrastrukturális elemekre épülő okosváros-szolgáltatásokat vizsgálni (Error Propagation Analysis, EPA).

A tanulmány példát mutat arra is, hogyan lehet a hibaterjedéshez szükséges információt mérnöki modellek segítségével leírni, így segítve a hibaterjedési vizsgálatok és a tényleges rendszer modelljének minél szorosabb kapcsolatát.

Emellett egy kiválasztott okos városi szolgáltatás, az intelligens kereszteződésekre építő, vezetési segítséget (driving assist) nyújtó szolgáltatások nemzetközi ajánlásain keresztül tekintjük át, hogyan használhatók a hibaterjedés-elemzési módszerek annak érdekében, hogy akár egy adott alkalmazás, akár egy rendszer kiértékelhető legyen, illetve mi ezen kiértékelés kapcsolata a rendszerrel szemben támasztott fejlesztési, ellenőrzési és üzemeltetési követelményekkel.

A Szerzők:

Prof. Dr. Jakab László, a BME Villamosmérnöki és Informatikai Karának egyetemi tanára, dékánja (2016–2019), az Okos város – okos közigazgatás kutatóműhely szakmai vezetője. Villamosmérnök diplomáját a BME-n szerezte meg 1981-ben. A műszaki tudomány kandidátusa 1992-ben, habilitált doktor 2013-ban, az MTA doktora címet pedig 2014-ben szerezte meg. Jelenleg az Elektronikai Technológia tanszék munkatársa.

Dr. Kocsis Imre, okl. mérnökinformatikus, a Budapesti Műszaki Egyetem Villamosmérnöki és Informatikai Kara Méréstechnikai és Információs Rendszerek Tanszékének adjunktusa. PhD-fokozatát a BME-en 2019-ben szerezte. Fő kutatási területe az elosztott megvalósítású informatikai szolgáltatások ellenállóképességre tervezése. Kutatásainak fő alkalmazási területei az „edge felhő” elemeket alkalmazó kiberfizikai rendszerek, illetve az elosztott főkönyvi rendszerek (Distributed Ledger Technology, DLT).

Dr. Gönczy László, okl. mérnökinformatikus és mérnök-közgazdász, a Budapesti Műszaki Egyetem Villamosmérnöki és Informatikai Kara Méréstechnikai és Információs Rendszerek Tanszékének adjunktusa. PhD-fokozatát a BME-en 2019-ben szerezte. Fő kutatási területe üzleti és kritikus IT-rendszerek tervezése és elemzése, különös tekintettel adatvezérelt módszerek alkalmazására teljesítmény és teljesítőképesség meghatározásában.

1. A HIBATERJEDÉSI ELEMZÉS JELENTŐSÉGE, SZABVÁNYOS FOLYAMATAI

A Kvalitatív módszerek és elemzési megközelítések felhasználása a kritikus infrastruktúrák védelmében és teljesítménymenedzsmentjében II. rész kismonográfia célközönsége az okos város és az e-közigazgatás fejlesztés témakörében érintett vagy a témakör iránt érdeklődő önkormányzati vezetők, képviselők és szakemberek, az állami tisztviselőkről szóló 2016. évi LII. törvény 1. § szerinti állami tisztviselők. Ezen felül anyagunk hasznos az egyetemi hallgatók, továbbá képzésükben, illetve továbbképzésükben részt vevő oktatók és a digitális technológia adta lehetőségek iránt nyitó kutatók-fejlesztők számára is. Célunk a tárgykörök tematikus, jól követhető módon való bemutatása.

„Smart" alkalmazások esetén, ideértve bármilyen olyan rendszert vagy rendszerelemet, amelyet „okos" melléknévvel látnak el a fejlesztők, illetve a szolgáltatók, különösen fontos a szolgáltatásbiztonság.

A rendszer kis hibáit ugyanis üzleti és fizikai szempontból nagyon felerősítheti az alkalmazási környezet. A „Smart" elem ezekben a rendszerekben alapvetően szoftveralapú. Az alárendelt érzékelő, végrehajtó elemekre és alrendszerekre azonban a „buta" alkalmazások évtizedes gyakorlatából van tapasztalat és metodika, illetve ezekre a szoftverekre vannak folyamatszabványok is.

A továbbiakban áttekintjük az ezzel kapcsolatos alapfogalmakat a szakirodalom alapján (Reliability–Availability–Maintainability). Először három olyan rendszertulajdonságot határozunk meg, amelyek a fejlesztők, üzemeltetők és nem utolsósorban a felhasználók számára is elsődleges fontosságúak.

Ezek a ***megbízhatóság***, a ***rendelkezésre állás*** és a ***karbantarthatóság*** (a Reliability-Availability-Maintainability angol szakterminológia kezdőbetűiből mozaikszóként RAM). Összességében ezek a rendszer által nyújtott szolgáltatások minőségét és ezen keresztül az életciklusköltségeit és alkalmazhatóságát befolyásolják.

Együttesen hatással vannak mind a termék, mind a rendszer életciklusára és használatára. A mai megbízhatósági tervezés eredete a második világháborúig követhető vissza.

A gyártási minőséggel kapcsolatos kérdések először az elektronikus és mechanikai alkatrészek területén keletkeztek. A mai tendenciák azonban az ipari és a fogyasztói termékek számának drámai emelkedésével és az ezekbe beépült integrált számítástechnikai funkciók következtében más területeket is érintenek. A fogyasztók, az ipari termelők és a szolgáltatók által egyaránt használt számítógépes rendszerek gyorsan növekvő integrációja miatt a szolgáltatásbiztonságot mind a hardver, mind a szoftver oldalán kiemelten kell kezelnünk.

A három fogalom kapcsolata erős. A meghibásodás fogalma a rendszer első rendellenes működésének időpontját, míg a karbantarthatóság a szolgáltatás helyreállását eredményező javítási folyamatot jellemzi.

A karbantarthatósági modellek érdekes kihívásokat is jelentenek. A javítás ideje a szervizbe szállításhoz, a hiba megállapításához, a rendszer javításához és összeszereléséhez szükséges idő összege, vagyis a javításra, az ellenőrzésre és visszaküldésre fordított teljes idő. Az adminisztratív késedelem (például ünnepek) szintén befolyásolhatja a javítási időket. Ezeknek az alfolyamatoknak gyakran van egy minimális technológia végrehajtási időtartamuk.

Ezáltal a meghibásodási küszöbérték-paraméterrel rendelkező karbantarthatósági modellezés meghatározott eloszlást eredményez. A küszöbérték-paraméter a javítás minimális legvalószínűbb ideje. A karbantarthatóság becslését tovább bonyolíthatja az egyes tényleges műveleteket megelőző várakozási sorban állás hatása. Ez a függőség gyakran kizárja az analitikus megoldást, és csak a szimuláció használata ad lehetőséget a karbantarthatóság jellemzésére.

A továbbiakban az alapvető meghatározásokat tárgyaljuk, röviden ismertetve a valószínűségi eloszlásukat, majd a RAM-tervezés (megbízhatóság, rendelkezésre állás és karbantarthatóság) szerepét a rendszerfejlesztés és -működés során.

Az ipari tömegtermékek esetében a megbízhatóság becslésének alapja a szóban forgó termék vagy hasonló technológiával előállított termékek meghibásodási statisztikája.

Az informatikai okos alkalmazások esetében különös kihívást jelent egyediségük, hiszen keverednek a tömegesen felhasznált termék jellegű elemek, például a futtató platformok, illetve az adott feladatra kidolgozott célmegoldások.

Ennek megfelelően, különösen abban az esetben, ha az alkalmazás kritikus, nemcsak egyszerű becslést szokás végezni, hanem külön vizsgálat tárgyává teszik a becslési bizonytalanság hatását.

Az első alapfogalom a ***megbízhatóság***. Ez annak a valószínűsége, hogy egy rendszer vagy rendszerelem az adott időtartam alatt meghatározott feltételek mellett meghibásodás nélkül teljesít. A pontos definíciónak természetes tartalmaznia kell a funkció, a környezet, az időskálák részletes leírását és a hibát is. Mindegyik elemet nehéz meghatározni, és annyit kell belőlük figyelembe venni, amennyit a valósághűség megkíván.

A ***karbantarthatóság*** vagy fenntarthatóság azt jelenti, hogy a rendszer vagy a rendszerelem meghatározott időn belül megjavítható egy meghatározott környezetben. A jobb karbantarthatóság rövidebb javítási időket jelent.

A ***rendelkezésre állás*** annak valószínűsége, hogy egy javítható rendszer vagy rendszerelem adott időpontban működőképes az adott környezeti feltételek mellett. Az elérhetőség a korábbiak szerint függ a megbízhatóságtól és a karbantarthatóságtól, mindezt részletesen tárgyaljuk később.

A meghibásodás az az esemény, amelyben egy rendszer bármely elemrésze nem, vagy nem megfelelően működik. A meghibásodási mechanizmus az a fizikai, kémiai, elektromos, termikus vagy egyéb folyamat, amely a hibát eredményezi. Számítógépes rendszerben a szoftver hibája is okozhat hibát, és a hibát megelőzhette egy hiba, amely egy elemen belül volt. A meghibásodási mód a mechanizmus útja vagy következménye, amelyen keresztül egy funkció vagy folyamat meghiúsul.

A megbízhatósági analízisben használt valószínűségi eloszlások közül a túlélési idő a komponens legalább t időpontig való hibamentes működését jellemzi. Ennek kiegészítője a t időpont előtti hiba valószínűsége. Ha egy T véletlen változót definiálunk, akkor a meghibásodás idejére

$$R(t) = R(T > t) = 1 - F(t)$$

ahol R (t) a megbízhatóság és F (t) a meghibásodási valószínűség. A meghibásodás valószínűsége a matematikai valószínűségeloszlás kumulatív eloszlásfüggvénye (CDF, cumulative distribution function). Az erre a célra használt folytonos eloszlások közé tartoznak az exponenciális, a Weibull, a log-normal és általános Gamma-eloszlások. A diszkrét eloszlásokat, mint például a Bernoulli, a binomiális és a Poisson, a hibák várható számának vagy a sikeres működés egyszeri valószínűségének a kiszámítására használják.

Ugyanazok a folyamatos matematikai megbízhatósági eloszlások is használhatók a karbantarthatóság és a rendelkezésre állásra modellezésére is, bár az értelmezésük más (például annak valószínűsége, hogy a hibás komponens a t idő alatt cserére vagy javításra kerül).

A rendelkezésre állásra vonatkozó becsléseknek azonban figyelembe kell venniük olyan folyamatokat is, mint például az adminisztratív késés, a szállítási idő, az alkatrész és a személyzet rendelkezésre állása.

A megbízhatósági és karbantarthatósági becslésben használt valószínűségi eloszlásokat modelleknek nevezzük, mivel csak becsléseket nyújtanak az értékelendő elemek valódi meghibásodására és javítására.

Ideális esetben az ilyen modellekben alkalmazott paraméterek értékeit az életciklus vagy a működési tapasztalat alapján becsülhetjük meg. Mindazonáltal költséges az ilyen közvetlen tesztek előzetes elvégzése vagy a rendszerek üzembe helyezése utáni hiteles működési adatok gyűjtése. Ezért gyakori a hasonló rendszerek, illetve a mérnöki legjobb gyakorlat adatainak felhasználása, de néha más módszereket is használnak. Ennek eredményeként a korlátozott adatokon alapuló becslések nagyon pontatlanok lehetnek. Az ilyen adatok összegyűjtésére szolgáló vizsgálati módszereket az alábbiakban tárgyaljuk.

A RAM (megbízhatóság, rendelkezésre állás és karbantarthatóság) tervezés során a rendszer összes, a fejlesztési életciklus során a szolgáltatásbiztonságot potenciálisan befolyásoló tulajdonságot figyelembe kell vennünk.

A megbízhatósági szabványok, a tankönyvek több fejlesztési folyamatmodellt is javasolnak. Az itt leírtak az Electronic Industry Association (EAI) és az USA kormányzata által közösen kifejlesztett és az USA Védelmi Minisztérium által elfogadott szabványra támaszkodnak, amely 4 alapvető folyamatot határoz meg:

1. a felhasználók igényeinek és az ebből származó kényszerek megértése,
2. a szolgáltatásbiztonságra tervezés,
3. a megbízhatóságra gyártás, valamint
4. a működés és a használat során történő felügyelet.

A felhasználói követelmények és az ebből származó kényszerek, korlátok megértésének folyamata magában foglalja a funkcionális követelményekre, a korlátozásokra (például tömegre, energiafogyasztásra, térbeli/használatból eredő „footprint”-re, lábnyomra, az életciklus költségére) vonatkozó információkat, valamint a RAM követelményeinek megfelelő igényeket.

Az ezekből kialakuló rendszerkövetelmények magukban foglalják a megbízhatóságra, a rendelkezésre állásra és a karbantarthatóságra vonatkozó elvárásokat a tervezett működési környezetben értelmezve. A RAM-követelmények meghatározása ugyan nagy kihívást jelent, de elengedhetetlen a fejlesztés sikeréhez, ugyanúgy, mint az általános funkcionális követelmények meghatározása.

A szolgáltatásbiztonságra tervezés során a felhasználói igényeknek megfelelő rendszertervezési alternatívákat kell, illetve lehet megfogalmazni, és műszaki-gazdasági szempontból kiértékelni. Ebben a fázisban a szolgáltatásbiztonságra tervezés célja a rendszer szolgáltatásbiztonságának a növelése olyan intézkedésekkel, mint a redundancia, a sokféle párhuzamos megoldásra törekvés, a beépített tesztlehetőségek, a fejlett (és így gyors) diagnosztikamódszerek és a modularitás, az alkatrész-, illetve részegységcsere lehetősége.

Ezen kívül a meghibásodások mértékének csökkentése olyan intézkedésekkel is lehetséges, mint például a nagyobb szilárdságú anyagok használata, a minőségi összetevők számának a növelése, a szélsőséges környezeti feltételek mérséklése vagy a karbantartási, ellenőrzési intervallumok lerövidítése.

A tervezési elemzések tartalmazhatnak mechanikai feszültség, korrózió- és sugárzáselemzéseket a mechanikai alkatrészekhez, mechanikai és elektromos alkatrészek hőelemzését, valamint elektromágneses interferencia (EMC) elemzéseket vagy méréseket az elektromos alkatrészek és alrendszerek számára.

A legtöbb számítógépes rendszerben a hardver elemek átlagos időtartama a meghibásodások között több százezer óra, így a legtöbb rendszer-tervezési intézkedés, a rendszer megbízhatóságának növelése a szoftverekre koncentrálódik.

A szoftverbiztonság növelésének legnyilvánvalóbb módja a minőség fokozása a gondos fejlesztési és tesztelési tevékenység révén. Ennek módjai a szoftverfejlesztés körébe tartoznak. Azonban a szoftver megbízhatóság és a rendelkezésre állás is növelhető az architektúra, vagyis a rendszer felépítés redundanciája, egymástól független és a sokféle rendszer elem révén. Ezt a redundanciát ki lehet egészíteni az adatkonzisztencia biztosításával, valamint a hibák felderítésével és a hibás részek kiiktatásával. A szoftver architektúrán belül az olyan intézkedések, mint a „watchdog” időzítők, amelyek a meghibásodások alatti időt mérik, vagy a folyamat kontroll, illetve az adatok integritásának ellenőrzése (például ciklikus redundancia ellenőrzések) növelhetik a megbízhatóságot.

Természetesen a dedikált szoftver mellett szolgáltatásokból építkező okos alkalmazások esetében is megvannak a rendszertechnikai megoldások pontos megfelelői. Például a kommunikáció esetében a párhuzamosan vagy külön-külön csatornákon keresztül működtetett adatátvitel alkalmas a teljes kommunikációs infrastruktúra kiesésének kompenzálására, legalábbis a kritikus funkciók esetében.

A rendszer RAM-jellemzőit a tervezés előrehaladtával folyamatosan értékelni kell. Ha a meghibásodási ráták ismeretlenek (mint az gyakori egyedi vagy egyedi fejlesztésű alkatrészek, szerelvények vagy szoftverek esetében), fejlesztési tesztelést végezhetnek az egyedi fejlesztésű alkatrészek megbízhatóságának értékelésére. A kvantitatív elemzéseken alapuló értékelések a rendszer kvantitatív megbízhatóságát és rendelkezésre állását értékelik. A formális analízis rendszerint a megbízhatósági blokkdiagramokon, ún. Markov-modelleken és Petri-hálókon alapulnak. A Markov-modellek és a Petri-hálók különösen hasznosak a redundanciát használó számítógépes rendszerek számára.

A minőségi elemzéseken alapuló értékelések az egypontos hibahelyekre (olyan egyszeres hibák, amelyek önmagukban képesek a szolgáltatás meghiúsítására), a hibák elszigetelésére, a helyreállítási metodikát és a karbantarthatóságot vizsgálják. Az elsődleges minőségi módszer a meghibásodási mód, hatás és kritikai elemzés (FMECA, lásd később).

A fejlesztési program során a diszkrepanciajelentést (DR) vagy a hibajelentést és a javító intézkedési rendszert (FRACAS) szintén fel kell használni azon hibamódok azonosítására, amelyeket az FMECA nem kezel. Azonosítani kell azokat a közös problémákat, amelyek javíthatók egy jobb tervezési vagy fejlesztési folyamaton keresztül.

A tervezési idő alatt keletkező egyéb elemzések szintén befolyásolják a RAM-ot. Az emberi tényezők elemzése szükséges annak biztosításához, hogy a kezelők és a karbantartók a rendszerrel olyan módon lépjenek kapcsolatba, amely minimalizálja a hibákat és a helyreállítási időket, amikor azok bekövetkeznek.

A RAM és a kiberbiztonsági rendszerek közötti is szoros kapcsolat áll fenn. Egyrészt a védelmi intézkedések csökkentik a rosszindulatú események miatti hibák gyakoriságát. Másrészt olyan eszközök, mint a tűzfalak, a házirend-végrehajtó eszközök és a

hozzáférés/hitelesítés (más néven „címtárkiszolgálók”) szintén válhatnak a rendszer megbízhatóságát és rendelkezésre állását csökkentő egyszeres hibaponttá vagy teljesítménybeli szűk keresztmetszetekké.

A megbízhatóságra a gyártás során a RAM-hoz kapcsolódóan számos gyártási probléma befolyásolja a minőséget.

- Ezek közül a legfontosabbak a gyártási folyamatok megismételhetőségének és egységességének biztosítása, valamint az ellátási lánc tételeinek egyértelmű meghatározása.
- Más problémák a gyártási, tárolási és szállítási tervekkel kapcsolatosak.
- A nagy, szoftverintenzív rendszereket érintik a konfigurációkezeléssel, az integrációs teszteléssel és a telepítéssel kapcsolatos problémák. Ide tartoznak a hibajelentő és korrekciós intézkedési rendszerek (PRACAS) vagy a FRACAS hibáinak vizsgálata és rögzítése a hibák kijavításával és a kijavítással kapcsolatos javításokkal kapcsolatban.
- Lényeges továbbá az ellátási lánc tételeinek egyértelmű meghatározása. A szervezeti szempontoktól függően ez lehet ugyanaz, amelyet a tervezés során használtak, vagy egy különálló rendszer.

A **„hibamód, hatás és kritikusság elemzés”** (FAILURE MODE, EFFECTS AND CRITICALITY ANALYSIS) egy, a meghibásodási mód és a hatáselemzés (FMEA) kiterjesztésén alapuló módszertan.

Az FMEA alulról felfelé irányuló, induktív, akár funkcionális, akár fizikai komponens szinten végrehajtható analitikai módszer. Az FMECA az FMEA-t kritikussági elemzéssel terjeszti ki, amely az egyes hibamódok előfordulási valószínűségéhez hozzáveszi a következményük súlyosságát is, így a kockázatelemzés egyszerű formája. Az eredmény kiemeli a viszonylag nagy kockázatú (például a gyakori vagy a súlyos következményekkel járó hibamódokat), és a korrekciós intézkedéseket a legkockázatosabbakra összpontosítja.

Az FMECA alapvető az olyan missziókritikus alkalmazásokban, mint az űrkutatás és a katonai alkalmazások, míg az FMEA különböző formái túlsúlyban vannak az olyan iparágakban, amelyekben a hibák potenciális hatása kevésbé kritikus.

Az FMECA-t eredetileg az 1940-es években dolgozta ki az amerikai hadsereg, amely 1949-ben kiadta a MIL-P-1629-et. Az 1960-as évek elején az Egyesült Államok Nemzeti Repüléstechnikai és Űrigazgatási Hivatala (NASA) különböző nevek alatt használja. 1966-ban a NASA kiadta az FMECA-eljárás leírását az Apollo programban való használatra. Ezt követően FMECA-t más NASA-programokban is használták (többek között a Vikingben, a Voyagerben, a Magellanban és a Galileóban).

A MIL-P-1629-et 1974-ben MIL-STD-1629 (SHIPS) váltotta fel, az FMECA fejlesztését néha helytelenül a NASA magának tulajdonítja. Az űrprogram fejlesztéseivel párhuzamosan az FMEA és az FMECA használata már elterjedt a polgári légiiparban is. 1967-ben a Society for Automotive Engineers (SAE) kiadta az első civil kiadványát az FMECA-nak. A polgári légiközlekedési ágazat jelenleg FMEA és a „Fault Tree Analysis” (Pilot 2002) kombinációját használja (a SAE ARP4761 útmutató szerint) az FMECA helyett, bár néhány helikoptergyártó továbbra is használja a FMECA-t a polgári célú járművek számára.

A Ford Motor Company az 1970-es években kezdte el használni az FMEA-t, a Pinto modelljével tapasztalt problémák hatására. Az FMEA az 1980-as évek során széles körben elterjedt az autóiparban. Európában a Nemzetközi Elektrotechnikai Bizottság 1985-ben tette közzé az IEC 812 (jelenleg IEC 60812) szabványt, amely mind az FMEA,

mind az FMECA általános célú felhasználásával foglalkozik. A British Standards Institute 1991-ben közzétette ugyanezen célból a BS 5760-5 szabványt.

1980-ban a MIL-STD-1629A váltotta fel mind a MIL-STD-1629, mind az 1977-es repüléstechnikai FMECA szabványt MIL-STD-2070. A MIL-STD-1629A-t 1998-ban utód nélkül törölték, de még ma is széles körben használják katonai és űrkutatási alkalmazásokhoz.

A módszertanban a különböző FMECA-szabványok között apró eltérések találhatók. A RAC CRTA (Reliability Analysis Center Critical Review and Technology Assessment) FMECA esetében az FMECA-analíziseljárás jellemzően a következő logikai lépésekből áll:

1. A rendszer meghatározása
2. A tervezést orientáló alapvető szabályok és feltételek definiálása
3. A rendszer blokkdiagramjainak összeállítása
4. A meghibásodási módok (komponens vagy funkcionális felbontású) azonosítása
5. Hibahatások/okok elemzése
6. Az eredmények visszacsatolása a tervezési folyamatba
7. A meghibásodások súlyosság szerinti osztályozása
8. A kritikusságra vonatkozó számítások
9. A kritikushiba-módok rangsorolása
10. A kritikus elemek meghatározása
11. Az eredmények visszacsatolása a tervezési folyamatba
12. A hibadetektálás, elszigetelés és javítás eszközeinek kiválasztása
13. Végezze el a karbantarthatósági elemzést
14. Az elemzés dokumentálása,
    a. a nem javítható tervezési területek (szolgáltatásbiztonságot sértő esetek),
    b. a kockázatcsökkentéshez szükséges különleges ellenőrzések
    c. opcionális javítási javaslatok
15. A korrekciós intézkedések végrehajtásának követése

Az FMECA egyaránt elvégezhető a funkcionális vagy fizikai részegység szintjén.

- A funkcionális FMECA a működési blokk szintjén a meghibásodások hatását veszi figyelembe, például egy tápegység vagy egy erősítő esetén.
- Az FMECA részegységek, az egyes komponensmeghibásodások, például ellenállások, tranzisztorok, mikroáramkörök vagy szelepek hatásait vizsgálja.

Az FMECA részegységekre, alkatrészekre vonatkozó része sokkal nagyobb erőfeszítést igényel, de előnyös a hiba-előfordulás valószínűségének jobb becslésére. Azonban a funkcionális FMEA sokkal korábban elvégezhető, segít a teljes kockázatértékelés jobb struktúrájának kialakításában, és másfajta betekintést nyújt a hiba következményeit redukáló alternatívák kidolgozásához. Az elemzések kiegészítik egymást.

A kritikussági elemzés mennyiségi vagy minőségi lehet, attól függően, hogy rendelkezésre állnak-e a támogató részhiba-statisztikák.

A rendszerleírás során az elemzendő legfontosabb rendszert definiáljuk és beillesztjük egy beillesztett hierarchiába, mint például rendszerek, alrendszerek vagy berendezések, egységek vagy részegységek és darabrészek.

Funkcionális leírások jönnek létre a rendszerek számára, és alrendszerekhez kerülnek hozzárendelésre, amelyek lefedhetik az összes működési módot és a felhasználói fázist.

- Mielőtt részletes elemzést folytatnánk, általában meghatározzuk és elfogadjuk az alapszabályokat és feltételeket. Ez magában foglalhatja például:
- Szabványosított felhasználási profilt meghatározott, rögzített időtartamú szakaszokkal
- A hibaarány és a meghibásodási mód adatainak forrását
- A rendszer beépített tesztje által észlelt hibaérzékelési lefedettséget
- Az elemzés funkcionális vagy részegységszintű lesz
- Figyelembe veendő kritériumok (felhasználhatóság, biztonság, karbantartás stb.)
- Rendszerleírás az alkatrészek vagy funkciók egyedi azonosításához
- Hibasúlyossági kategóriák meghatározása

Ezután a rendszerek és alrendszerek funkcionális blokkdiagramokban kerülnek ábrázolásra. A megbízhatósági blokkdiagramokat vagy hibafákat (Pilot 2002) általában ugyanabban az időben állítják össze. Ezeket az ábrákat a rendszerhierarchia különböző szintjein lévő információáramlás nyomon követésére, a kritikus utak és interfészek azonosítására és az alacsonyabb szintű hibák magasabb szintű hatásainak azonosítására használják.

Az egyes elemrészek vagy az elemzés által lefedett minden egyes funkció esetében ki kell dolgozni a hibamódok teljes listáját. A funkcionális FMECA esetében a tipikus hibamódok a következők:

- Nem ütemezett működés
- Nem működik, ha szükséges
- A kimeneti adatok elvesztése
- Időszakos kimenetek
- Hibás kimeneti adatok (az aktuális állapot miatt)
- Érvénytelen kimeneti adatok (bármilyen feltétel esetén)

Az FMECA-darabrész esetében hibaállapot-adatok származhatnak olyan adatbázisokból, mint a RAC FMD-91 vagy a RAC FMD-97 ajánlások. Ezek az adatbázisok nemcsak a meghibásodási módokat, hanem a meghibásodási módarányokat is tartalmazzák.

Minden egyes funkciót vagy darabot mátrix formában sorolunk fel, sorrendben egy-egy sorral. Mivel az FMECA rendszerint igen nagy adatkészleteket tartalmaz, minden tételhez (funkcióhoz vagy darabrészhez) és minden tétel minden meghibásodási módjához egyedi azonosítót kell rendelni.

A hibahatásokat az FMECA-mátrix minden egyes sorában meg kell határozni és be kell írni, figyelembe véve az alapszabályokban meghatározott kritériumokat. A hatásokat külön leírjuk a helyi, a következő magasabb és a vég (rendszer) szinten. A rendszerszintű hatások a következők lehetnek:

- Rendszerhiba
- Megsemmisített működés
- Rendszerállapot-hiba
- Nincs közvetlen hatás

A különböző hierarchikus szinteken használt hibahatási kategóriákat az elemző mérnökök ítélik meg.

A súlyossági besorolást az egyes egyedi elemek minden meghibásodási módjához hozzárendelik, és az FMECA-mátrixra kerülnek, rendszerszintű következmények alapján. A besorolások egy kis csoportját használják, általában 3-10 súlyossági szinttel.

Súlyossági kategóriák (MIL-STD-882) lehetnek a „Végzetes", úgymint a halál, végleges teljes fogyatékosság, a veszteség meghaladhatja az 1 millió $-t, vagy visszafordíthatatlan súlyos környezeti károkat okozhat, vagy amely sérti a törvényt vagy a szabályozást.

„Kritikus", ami végleges részleges fogyatékosságot, sérüléseket vagy foglalkozási megbetegedést eredményezhet, amely legalább három személy kórházi ellátását eredményezheti, a veszteség meghaladhatja a 200 000 dollárt, de kevesebb, mint 1 millió dollár, vagy visszafordíthatatlan környezeti károkat okozhat a jog vagy szabályozás megsértése miatt.

Lehet „Marginális", ami sérülést vagy foglalkozási megbetegedést eredményezhet, amely egy vagy több elveszett munkanapot jelent, a veszteség meghaladja a 10 000 dollárt, de kevesebb, mint 200 000 dollár, vagy mérsékelt környezeti károkat eredményez anélkül, hogy megsértené a jogszabályokat vagy a szabályozást, amennyiben a helyreállítási tevékenységek megvalósíthatók.

„Elhanyagolható" súlyosság, ami sérülést vagy betegséget okozhat, ami nem eredményez elveszett munkanapot, a veszteség meghaladja a 2 ezer $-t, de kevesebb, mint 10 ezer $, vagy minimális környezeti károkat, amelyek nem sérti a törvényt vagy a szabályozást.

Hibafelderítési módszerek alkalmazásával minden egyes elem és meghibásodási mód esetében a rendszer képes észlelni és jelenteni a szóban forgó hibát. Az FMECA-mátrix minden sorában az alábbiak valamelyike szerepel:

- Normál: a rendszer helyesen jelzi a személyzet biztonságos állapotát.
- Rendellenes: a rendszer helyesen jelzi a személyzet közbeavatkozását igénylő hibát.
- Helytelen: a rendszer hibásan jelzi a biztonságos állapotot hiba esetén, vagy figyelmezteti a személyzetet, hogy nem létezik hiba (hamis riasztás).

A hibamód kritikusságának értékelése minőségi vagy mennyiségi lehet. A minőségi értékeléshez egy valószínűségi kódot vagy számot kell hozzárendelni és beírni a mátrixba. Például a MIL-STD-882 öt valószínűségi szintet használ:

- „Gyakori", valószínűleg gyakran fordul elő az elem élettartama során, vagy folyamatosan tapasztalt.
- „Valószínű", többször vagy gyakran fordul elő.
- „Alkalmi", valószínűleg bizonyos elemek életében előfordulhat, vagy többször fordul elő.
- „Távoli", nem valószínű, de előfordulhat egy elem életében, nem valószínű, de észszerűen várható, hogy előfordul.
- „Valószínűtlen", valószínűtlen, feltételezhető, hogy előfordulhat, valószínű, hogy nem tapasztalható, nem valószínű, hogy előfordul, de lehetséges.

A meghibásodási módot ezután egy kritikus mátrixon lehet ábrázolni: súlyossági kódként, mint az egyik tengely, és valószínűségi szint kódként, mint a másik. A kritikus értékek kiszámítása a következő értékekkel történik: alaphibaarány, hibamódarány, valószínűség, illetve felhasználási fázis időtartama. A kritikus érték egy-egy hibára ezeknek az értékeknek a szorzata. A teljes rendszer jellemzéséhez a kritikus értékeket összeadjuk.

Az alaphibaarányt általában a MIL-HDBK-217, a PRISM, a RIAC 217Plus vagy egy hasonló modellen alapuló hibaarány-előrejelzés alapján táplálják az FMECA-ba. A meghibásodási mód aránya egy adatbázisforrásból, például RAC FMD-97-ből származhat. FMECA funkcionális szinthez műszaki mérésekre lehet szükség a hibamódarány meghatározásához. A feltételes valószínűségi szám azt a feltételes valószínűséget jelenti, hogy a hibahatás az azonosított súlyossági osztályozást eredményezi, mivel a meghibásodási mód bekövetkezik. Ez azt jelenti, hogy az elemző megítélése szerint a veszteség valószínűsíthető. A grafikus analízishez a kritikus mátrixot lehet ábrázolni (a kritikus értéket egy tengelyen, és a súlyossági kódot a másikon alkalmazva).

Miután az egyes elemek egyes meghibásodási módjaira vonatkozóan elvégezték a kritikai értékelést, az FMECA-mátrix a súlyosság, a minőségi valószínűségi szint vagy a mennyiségi kritikusság sorrendje szerint válogatható. Ez lehetővé teszi, hogy az elemzés azonosítsa a kritikus elemeket és a kritikus hibamódokat, amelyekre a tervezési során ki kell térni.

Az FMECA elvégzése után javaslatokat tesznek a kritikus hibák következményeinek csökkentésére. Ez magában foglalhatja a nagyobb megbízhatóságú komponensek kiválasztását, csökkentve a kritikus elemek stresszszintjét, vagy redundanciát, vagy további hibaérzékelési lehetőségeket adhatunk a rendszerhez.

Az FMECA általában a karbantarthatósági elemzéshez, a karbantartás-elemzéshez és a logisztikai támogatás elemzéséhez is csatlakozik, amelyek mindegyike az FMECA adatait használhatja.

Az FMECA-jelentés a rendszer leírását, alapszabályait és feltételezéseit, következtetéseit és ajánlásait, a nyomon követhető korrekciós intézkedéseket és a mellékelt FMECA-mátrixot tartalmazza, amely lehet táblázatkezelő, munkalap vagy adatbázis formában.

Kockázatprioritási számítás során a RAC CRTA-FMECA és a MIL-HDBK-338 egyaránt a kritikus elemzés alternatív módszere szerint azonosítja a kockázati prioritási számot (RPN). Az RPN a detektálhatóság (D) x súlyossága (S) x előfordulásának (O) szorzatának eredménye. Minden egyes skálán 1-től 10-ig, a legmagasabb RPN 10x10x10 = 1000. Ez azt jelenti, hogy ez a hiba nem mutatható ki ellenőrzéssel, hatása nagyon súlyos és az előfordulás szinte biztos. Ha az esemény nagyon ritka, akkor ez lenne 1, és az RPN értéke 100-ra csökken. Tehát a kritikus elemzés lehetővé teszi a legnagyobb kockázatokra való összpontosítást.

Az FMECA erősségei magukban foglalják annak átfogó jellegét, a meghibásodás okai és hatásai összefüggéseinek rendszeres kialakítását, valamint annak a képességét, hogy a tervezési korrekciós intézkedések egyedi hibamódjait mutassák be. A gyengeségek közé tartozik a kiterjedt munkaigény, a számottevő triviális esetek száma, valamint a többszörös meghibásodási forgatókönyvek vagy a nem tervezett keresztrendszer-hatások, például a beágyazott áramkörök kezelésére való képtelenség.

Egy FAA (Federal Aviation Administration) kutatási jelentés szerint a kereskedelmi űrszállítás esetében a hibamódok, hatások és a kritikus elemzés kiváló veszélyelemzés és kockázatértékelő eszköz, de más korlátokkal számolnunk kell. Ez az alternatíva nem veszi figyelembe a kombinált meghibásodásokat, vagy jellemzően a szoftver és az emberi interakció szempontjait. Általában optimista becslést ad a megbízhatóságról. Ezért a FMECA-t más analitikai eszközökkel együtt kell használni a megbízhatósági becslések meghatározása során.

Az IEEE 1044 (IEEE 1044-2009) a szoftverhibák súlyosság szerinti besorolását adja, megjegyezve, hogy sok más területen, például nukleáris technika, vannak hasonlóak (például International Nuclear Event Scale). Kiemelendő, hogy miután nincs hibamentes szoftver, és a hatások erősítése miatt a kis szoftverhiba is okozhat nagy rendszerhibát, a vezérlő szoftver hibaskáláján egy kis hiba is okozhat alkalmazás szintjén súlyosat, és a hibaterjedési analízis célja éppen ennek az erősítésnek blokkolása.

Ez a szabvány összefoglalja a hibák és a hibák osztályozásához szükséges tulajdonságok alapvető készletét. Felismerték, hogy vannak olyan hibák vagy hibák egyéb tulajdonságai, amelyek egyedi értéket képviselnek bizonyos alkalmazásokban vagy üzleti követelményekben. Ez a szabvány minden szoftverre (beleértve az operációs rendszereket, az adatbáziskezelő rendszereket, az alkalmazásokat, a tesztrendszert, a firmware-t és a beágyazott szoftvert) és a projekt, termék vagy rendszer életciklusának bármely szakaszára vonatkozik, amelyekkel a szoftver kifejlesztése, működtetése és fenntartása során találkozhatunk. A besorolási szempontokat nem befolyásolja az életciklusmodell kiválasztása, és ez a választás kívül esik e standard hatálya alól. A kiválasztott életciklusون alapuló besorolási értékek egyéni igazítása várható, és összhangban van a jelen standard szándékával.

Ennek a szabványnak a célja olyan közös szókincs definiálása, amellyel a különböző emberek és szervezetek hatékonyan tudnak kommunikálni a szoftveres anomáliákkal kapcsolatban, és olyan közös tulajdonságkészletet hoznak létre, amely támogatja az ipari technikákat a szoftverhibák és hibaadatok elemzéséhez.

A szervezet az osztályozás folyamatát az alábbiak szerint határozza meg:

a) A hibák és meghibásodások osztályozásával elérni kívánt cél (ok).
b) A felhasznált referenciaszabvány (például a specifikációban, szerződésben vagy tervben leírtak szerint), amely segítségével meghatározzuk, hogy melyik rendszer/szoftver viselkedése hibás.
c) Hogyan oldjuk meg az osztályozási döntésekkel kapcsolatos véleménykülönbségeket vagy konfliktusokat.
d) Mikor kezdődik, illetve fejeződik be az osztályozás a projekt vagy a termék életciklusán belül.
e) A projekt-, termék- vagy szervezetspecifikus értékek, amelyek alkalmasak a besorolási értékek hozzárendelésére.
f) Ki a felelős a besorolási értékek meghatározásáért, minden hiba és meghatározás tekintetében.
g) Hol és hogyan kell fenntartani, tárolni, őrizni és karbantartani a besorolási adatokat.

A szervezetnek a hibakategóriákra vonatkozó értékeket kell rögzítenie. A szabványban összefoglalt a tulajdonságkészlet természetesen nem teljes. A kiválasztott attribútumok mintaértékére javaslatot tesz, de ezek az értékek csak tájékoztató jellegűek. Az attribútumok értékei valószínűleg folyamatosan kerülnek rögzítésre, amikor a szervezet a hibát orvosolja. A hibaállapot státusza lehet berögzített, észlelt vagy eltávolított, azonban a hibaállapot helyett gyakori a kapcsolódó hibajelentés állapotának nyomon követése, ezért az állapothoz rendelt érték jellemzően a szervezet hibakezelési folyamatához kapcsolódó munkafolyamat állapotára utalhat. Nincsenek kötelező statisztikai értékek, mivel számos szervezetspecifikus hibakezelési folyamat létezik. A hibakezelés folyamata során használható jellemzőket az alábbi táblázat mutatja be:

Jellemző	Definíció
Defect ID	A hiba egyedi azonosítója.
Leírás	A hiányzó, a rossz vagy a felesleges leírása.
Állapot	Aktuális állapot a hibajelentés életciklusán belül.
Eszköz	A hibát tartalmazó szoftvereszköz (termék, összetevő, modul stb.).
Artifact	A hibát tartalmazó speciális szoftvermunkaeszköz.
Észlelt verzió	A szoftververzió azonosítása, amelyben a hibát észlelték.
Korrigált verzió	Azon szoftververzió azonosítása, amelyben a hibát korrigálták.
Elsőbbségi	A feldolgozás rangsorolása, amelyet a hibaértékelésért és megszüntetéséért felelős szervezet végez az egyéb bejelentett hibákhoz kapcsolódóan.
Súlyosság	A legmagasabb hibahatás, amelyet a hiba okozhatott (vagy nem), ahogy azt a szoftverfejlesztésért felelős szervezet meghatározza.
Valószínűség	A hiba ismétlődésének valószínűsége.
Hatás	A követelménykategória, amelyet egy hiba okozta meghibásodás okoz.
Típus	A besorolás a kód azon osztályán alapul, amelyen belül a hiba megtalálható.
Mód	A kategorizálás attól függően, hogy a hiba a helytelen végrehajtásnak vagy reprezentációnak, valami nem szükséges hozzáadásnak vagy mulasztásnak tulajdonítható-e.
Hibagenerálási aktivitás	Az a tevékenység, amely során a hibát létrehozták/ beillesztették (azaz, amiből a hibát tartalmazó rész származik).
Érzékelési tevékenység	Az a tevékenység, amely során a hibát észlelték (például ellenőrzés vagy vizsgálat).
Meghibásodás azonosítója	A hiba által okozott meghibásodás azonosítója.
Változáshivatkozás	Azonosító a kezdeményezett korrekciós változtatási kérelemhez, a hiba javítása érdekében.
Diszpozíció	A hibajelentés végleges elhelyezése a lezárás után.

1. táblázat. Hibakezelés folyamatának jellemzői
Forrás: a szerzők saját szerkesztése az IEEE 1044 alapján

A hiba osztályozása során a szervezetnek az összes hibatulajdonság-értéket rögzítenie kell. Az alábbi táblázatban felsorolt jellemzők csak tájékoztató jellegűek.

Jellemző	Definíció
Hibaazonosító	A hiba egyedi azonosítója.
Állapot	Aktuális állapot a hibajelentés életciklusán belül.
Cím	A meghibásodás rövid leírása az összefoglaló jelentéskészítés segítéséhez.
Leírás	Az anomáliaviselkedés és a feltételek teljes körű leírása, beleértve a hibát megelőző események és/vagy felhasználói műveletek sorrendjét.
Környezet	A működési környezet azonosítása, amelyben a meghibásodást megfigyelték.
Konfiguráció	A konfigurációs helyzet részletei, beleértve az érintett termék- és verzióazonosítót.
Súlyosság	A szoftverfejlesztésért felelős szervezet perspektívájából meghatározva.
Elemzés	Az ok-okozati elemzés végső eredményei a meghibásodásvizsgálat befejezésekor.
Diszpozíció	A hibajelentés végleges elrendezése.
A megfigyelő személye	Az észlelő személy, aki figyelte a hibajelenséget (és akitől további részletek nyerhetők).
Hibajelentést megnyitotta	A személy, aki megnyitotta (benyújtotta) a hibajelentést.
Hozzárendelt	A személy, akit a szervezet rendelt, hogy vizsgálja meg a hiba okát.
Lezáró	A személy, aki lezárta a hibajelentést.
Megfigyelés dátuma	A hibajelenség időpontja/ideje.
Megnyitás dátuma	A hibajelentés megnyitása (benyújtva).
Dátum lezárva	A hibajelentés lezárásának dátuma és időpontja, a végleges rendezést rendelik hozzá.
Tesztreferencia	A hiba bekövetkezésekor elvégzett specifikus vizsgálat azonosítása (ha van ilyen).
Eseményreferencia	A kapcsolódó esemény azonosítása, ha a hibajelentést a szerviz vagy a help desk hívása/kapcsolata generálta.
Hibakeresés	A hiba azonosítása, hogy mi a hiba oka.
Hibajelentés	Kapcsolódó hibajelentés azonosítása.

2. táblázat. A hibák osztályozása
Forrás: a szerzők saját szerkesztése az IEEE 1044 alapján

Az IEEE 1633 (Ieee 1633-2016) „A szoftver megbízhatóság ajánlott gyakorlati módszerei” szabvány szerint a szoftver alapvetően rugalmas és könnyen kezelhető. Ez azt is jelenti, hogy többféleképpen lehet előidézni a hibajelenségeket, szándékosan és szándékolatlanul. A szoftverhiba megjelenése potenciális hibát okozhat. Sok esetben a szoftverhibákból eredő hibák kiszámíthatók és elkerülhetők, de a hibák a következők miatt is előfordulhatnak:

1. A rendelkezésre álló idő/erőforrások hiánya ahhoz, hogy megtaláljuk az összes olyan szoftverhibát, amely hibahatást okozhat.
2. Rendkívül összetett eseményvezérelt rendszerek készülhetnek, amelyeket nehéz megtervezni, és ezért megvalósítani és tesztelni.
3. Olyan a szervezeti kultúra, ahol nincs meg az elégséges szigorúság, és nincsenek készségek vagy módszerek a hibák megállapításához.
4. Olyan technikai döntések születnek, amelyek helytelen architektúrát vagy tervezési döntést eredményeznek, és nem támogatják az érdekelt felek specifikációit.
5. Nem megfelelő projekt vagy kockázatkezelés, amely késleltetéshez vezet, ami kevesebb megbízhatósági tesztet eredményez.
6. Működési szerződéses kérdések, a rossz előírások és az érdekelt felek közötti kommunikáció miatt kialakuló hiányos interoperabilitás.

Még néhány kisebb szoftverhiba is vezethet monetáris katasztrófákhoz, például egy törölt projekthez. A hardveres (HW) hibák véletlenszerűek lehetnek, az elhasználódásnak vagy a szisztematikus tervezési hibának köszönhetően. A megbízhatóság karbantarthatóságát (RMA) a hardverhibák megelőzésére és kezelésére használják. A szoftverhibák a követelmények, tervek, kódok vagy interfészek rendszeres hibáiból adódhatnak. Ezért a szoftveres meghibásodás nem igényel RMA-t, hanem helyreállítási lépést egy meglévő telepítéshez. A szoftverhibák gyakori hibák lehetnek, mivel ugyanaz a meghibásodási mód több hibát okozhat a szoftver egynél több részében.

Az IEEE 1633 szabványban és ennek utódaiban ajánlott gyakorlatnak az a célja, hogy a szervezet foglalkozzon a szoftver megbízhatóságával (SR). Nem foglalkozik kifejezetten a rendszerek megbízhatóságával, szoftverbiztonságával vagy kiberbiztonságával. Ugyanakkor hangsúlyozza, hogy a biztonsági és kiberbiztonsági követelmények a kezdeti kockázatértékelés részét képezik. Az ajánlott gyakorlat csak röviden foglalkozik a szoftverek minőségével. Ez az ajánlott gyakorlat közös kiindulópontot biztosíthat a vitához, és előírja a szoftver megbízhatóságának értékelésére és előrejelzésére szolgáló módszereket. Az ajánlott gyakorlat célja a szoftverek tervezésének, fejlesztésének és tesztelésének támogatása, valamint olyan megalapozás létrehozása, amelyen a szakemberek egységes módszereket dolgozhatnak ki a szoftver megbízhatóságának értékelésére. Célja, hogy megfeleljen a szoftverszakemberek és a felhasználók igényeinek, akik szembesülnek a különböző terminológiával a megbízhatóság mérése és a multimodellezés és adatgyűjtési módszerek területén. Ez az ajánlott gyakorlat tartalmazza az SR-mérés alkalmazásához szükséges információkat egy-egy projekthez. Ez magában foglalja a szoftveréletciklus (SLC) SR-tevékenységét a követelmények generálásából kiindulva az alkalmazás azonosításával, a követelmények meghatározásával és elemzésével, valamint a megvalósítás folytatásával.

Ez a szabvány a következőket tartalmazza:

- Közös terminológia.
- A szoftvert vagy a projektet érintő szoftverbiztonsági kockázatok értékelése.
- Szoftverhiba módú elemzések, amelyek segíthetnek azon hibák típusának azonosításában és csökkentésében, amelyek leginkább a rendszer meghibásodásához vezetnek.
- Modellek a szoftverfejlesztés korai fejlődésének előrejelzésére.
- Modellek a szoftver megbízhatóságának tesztelésére és működésére.
- A teszt lefedettsége és a teszt kiválasztása.
- Adatgyűjtési eljárások az SR becslés és előrejelzés támogatására
- A szoftver kiadásához vagy korrekciójához szükséges feltételek meghatározása.
- Azon elemek azonosítása a szoftverrendszerben, amelyek vezető szerepet töltenek be az újratervezésben a megbízhatóság javítása érdekében.

A szoftvermegbízhatóságra tervezés (SRE, Software Reliability Engineering) megalapozott tudományterület, amely segítheti a szervezeteket termékeik és folyamataik megbízhatóságának javításában. Fontos, hogy egy szervezetnél folyamatfegyelem legyen, ha nagy megbízhatóságú szoftvert állít elő. A szabványban konkrét gyakorlatok és ajánlások találhatók, amelyek mindegyike összefüggésben van a szoftverfejlesztési életciklussal. Egy adott gyakorlat megvalósítható vagy alkalmazható az életciklus egy adott szakaszában, vagy több szakaszban is alkalmazható. A projekt kezdetétől a végéig változnak a szükséges SRE-folyamatok. Az egyes SRE-gyakorlatok különböző mértékben kerülnek végrehajtásra a fejlesztés vagy a művelet minden egyes fázisában. Például szoftverfejlesztési projektekben a hibamódok és hatáselemzések (FMEA) tipikusan az életciklus elején kerülnek végrehajtásra.

Az SRE lehetővé teszi a fejlesztőknek, hogy bízzanak abban, hogy egy szoftverrendszer megfelel a mennyiségi és minőségi megbízhatósági követelményeknek, vagy meghaladja azokat. Az SRE egy adatközpontú stratégiát alkalmaz ennek elérése érdekében, amely magában foglalja a következő műveleti területeket, elemzéseket és mutatókat (de nem korlátozódik azokra):

- Tervezés: Meghatározza a szoftvermegbízhatóság (SR) program hatókörét a rendelkezésre állás alapján az erőforrások és a projekt igényeinek megfelelően. Meghatározza a szoftvercserére cserélhető egységeket (LRU), amelyek bekerülnek az SRE-be.
- A szoftverrel kapcsolatos kockázatok elemzése és felmérése és a hibaüzenetek korai felmérése, olyan követelmények és tervek kidolgozása, amelyek megbízhatóbb szoftvereket támogatnak.
- Predikció – Becslések elvégzése a rendszer működési profiljának (OP) megfelelően, az érzékenységi elemzés elején végrehajtva. A megfelelő célok azonosítása és a szoftver megbízhatóságát befolyásoló tervezett fejlesztési tevékenységek gyengeségeinek azonosítása. A követelmények meghatározása az elfogadható meghibásodási ráta, a maradék hibák, a hibák elmaradása, a rendelkezésre állás vagy más mennyiségi szempontok alapján, figyelembe véve a kritikusságot és a felületet, valamint az architektúra-függőségeket.
- a terepi felhasználás szempontjából reprezentatív tesztelés-magatartás tesztelés, a kódot a szoftvert OP-jének megfelelően teszteljük. Fejleszteni kell a kísérleti részegységeket, amelyek elérik reprezentatív mintavételt a műveletekről, módokról és esetleges program ütközésekről. A hibák számának meghatározása (minden egyes esemény hiba), és korrigálva, és figyelemmel kísérve azon a

felfedezési arányát. Szükség szerint felül kell vizsgálni a becsléseket. A rendszer működési megbízhatóságának becslése a megfigyelt meghibásodások és egy megbízhatósági növekedési modell vagy hibaintenzitási grafikon alkalmazásával. Meghatározása az elfogadható maradék hibák becsült számának.
- Kibocsátás - az SRE-t használjuk a kibocsátási döntés támogatására. Elemezzük a hibák és a támogatási hibák alakulását, ez átmenet a szoftver mérföldkövekből. Dokumentáljuk a szoftver kibocsátási készenlétét, a becsült működési megbízhatósággal és a maradék hibák becsült számával.
- Működés - figyelemmel kísérése és a szoftver hibák számának és ütemének modellezése a javítás érdekében, jövőbeli SR elemzések.

Az (Ieee 1633-2016) irodalom a szoftver megbízhatósági tervezés hasznos összefoglalását adja. Megbízhatósági tervezés már több mint 50 éve létezik. Alapvető előfeltétel szinte minden modern rendszer számára. Az elmúlt évtizedek során sok elmélet született, de az ilyen modellek alkalmazásának gyakorlati útmutatásai erősen hiányoznak. Az érdekelt felek sokfélesége pragmatikus útmutatást és eszközöket igényelhetnek a szoftvermegbízhatósági modellek alkalmazásához, a valódi szoftverek vagy firmware projektek értékeléséhez a szoftverfejlesztési életciklus minden szakaszában. A megbízhatósági mérnököknek hiányozhatnak a szoftverfejlesztési tapasztalatok és előfordulhat, hogy a szoftvermérnök nem ismeri a szoftver megbízhatóságának előrejelzésére szolgáló módszereket. Mindkettő számára lehetnek kihívások az adatok elemzéséhez szükséges adatok megszerzéséhez. A mai modern rendszerek ritkán csak egyetlen szoftverkomponenst tartalmaznak. Ezeknek gyakran több tucatja van: ügyfélszoftver alkalmazások, COTS (kereskedelmi off-the-shelf szoftver), FOSS (szabad és nyílt forrású szoftver), Firmware, Middleware, Glueware, Operációs rendszer

A szoftver hibák és a kritikusságuk mértékének meghatározásához, a szoftver hibáinak pontos mérése érdekében ezeket projekt-specifikus alapon kell meghatározni. A hiba meghatározását és a pontozási kritériumokat elő kell készíteni bármely specifikáció kidolgozása előtt. A szoftverhibákat az alrendszer vagy a rendszer szintjén kell meghatározni. Egy hibát, vagy eseményt a szoftver vagy a hardver is okozhatja, és az esemény leírása nem lehet szubjektív.

Értékelnünk kell az adatgyűjtési rendszert, mert a szoftver megbízhatósági modelljének használata előtt tudnunk kell, hogy milyen adatok állnak rendelkezésre. Nincs feltétlenül minden felhasználónak közvetlen hozzáférése a tesztelés során keletkező hibaadatokhoz, míg a fejlesztésben részt vevő személyek nem mindig férhetnek hozzá a terepi hibaadatokhoz. A régebbi SRE útmutatók azt feltételezték, hogy a felhasználó tökéletes adatokat kap, például a szoftverhibák közötti másodpercek számát, de feltételeznünk kell, hogy az adatok tökéletlenek, a hibák közötti pontos időtartam gyakran nem ismert, és bár az adatok bizonyos esetekben hasonlítanak, valószínűleg nem illeszkednek tökéletesen az aktuális projekthez.

Az SRE-hez szükséges eszközök vonatkozásában tudomásul kell vennünk, hogy nincs egyetlen eszköz, amely lefedi az összes SRE-feladatot, vagyis több eszközre lehet szükségünk. Ilyenek lehetnek:
- Gyűjtés, szűrés, hibaadatok szervezése
- A szoftver méretének előrejelzése és mérése
- Hibák és hibák előrejelzése
- Az SWR növekedésének előrejelzése a vizsgálat és a működés során
- Hibamódok elemzése

Fejlesztenünk kell folyamatosan az SR (Software Reliability)-tervet. Az SR-terv biztosítja, hogy az SRE-feladatokat a projekt kezdetén határozzák meg és nyomon kövessék a projekt során. A terv állhat önállóan, vagy képezheti akár a megbízhatósági, akár a szoftverfejlesztési terv részét. A szoftverhiba gyökérokozati elemzés (RCA, Root Cause Analysis) biztosítja, hogy az SRE fejlesztési erőfeszítései a hibák megfelelő típusaira vonatkozzanak. Ha a hibák többségét azonosítjuk a tervezési fázisban, akkor nem szükséges az összes SRE-erőfeszítést a kódolási gyakorlatok javítására fordítani. A szoftver megbízhatóságának felmérése bizonyos hiányosságokat és erősségeket állapíthat meg, amelyek bizonyos hibák „kiterjedéséhez" vezethetnek. A gyökérokozati elemzés megállapíthatja a hibák „típusát", például: hibás követelmények, hibás kivitel, hibás végrehajtás, hibás interfészek, hibás változások vagy javító intézkedések, hibás forrás és verziókezelés, hibás telepítési csomag, hibás használhatóság.

Ezek minden konfigurációs tétel esetében egyediek lehetnek, és azok akkor is, ha azonos fejlesztési folyamatokkal rendelkeznek. A hibák a rossz követelményekből, a rossz tervezésből, a rossz kódolási gyakorlatokból vagy a rossz változásszabályozásból is eredhetnek.

A szoftver-FMEA a hardver-FMEA-hoz hasonlóan zajlik. A szoftver és hardver FMEA közötti különbség, hogy nyolc szoftveres terület van, beleértve a

- követelmények meghatározását,
- a tervezést,
- az interfész kialakítását,
- a javító intézkedéseket/változtatásokat,
- használhatóságot,
- szervizelhetőséget.
- a sebezhetőséget
- és a szoftver előállítását.

Több mint 400 meghibásodási mód/kiváltó ok van, amelyek legalábbis relevánsak az összes szoftverrendszer szempontjából.

2.

2. A HIBATERJEDÉS-ELEMZÉS ALAPFOGALMAI

A számítástechnikai eszközökkel megvalósított rendszerek *szolgáltatásbiztonsági* (dependability) jellemzői a rendszer specifikációjának részét – a funkcionális megfelelőségen túlmutató, azaz nem „funkcionális” vagy „extra-funkcionális” részét – képezik. A szolgáltatásbiztos számítástechnika szakmai konszenzusa (Avizienis et al. 2004) ezen jellemzők közé érti a *rendelkezésre állást* (availability), a *megbízhatóságot* (reliability), a *biztonságosságot* (safety), az *integritást* (integrity) és a *karbantarthatóságot* (maintainability).

Ezen attribútumok megkövetelt és működés közben elvárt számszerű jellemzőinek (metrikáinak) teljesülését egy valamilyen szempontból kritikus alkalmazás – vagy informálisan, de talán találóan fogalmazva: „elég fontos” alkalmazás – esetén a megfelelő tervezés, a rendszerbe beépített mechanizmusok és az üzemeltetés eljárásrendjei együttesen biztosítják. Különösen fontos kategóriát alkotnak a *biztonságkritikus* (safety critical) alkalmazások, mint például a vasúti rendszerek; ezeknél elsődleges szempont az emberéletet követelő balesetek valószínűségének bizonyítottan elégségesen alacsony volta. Más rendszereknél a *szolgáltatásbiztonságra tervezés* (design for dependability) célja alapvetően üzleti; egy üzletmenet-kritikus (business critical) alkalmazás esetén a szervezeti kockázatok kezelése szempontjából kiemelt faktor, hogy mik a szolgáltatás rendelkezésre állásának, megbízhatóságának és integritásának számszerű jellemzői.

2.1. Hibaok, hibás állapot, hibahatás

A szolgáltatásbiztonság elemzésében és tervezésében a (funkcionális) specifikációtól való eltérés jellemzését három alapfogalom segíti: a hibaok (fault), a hibás állapot (error) és a hibahatás (failure). A hibahatás maga az elvárt működéstől való eltérés ténye, illetve eseménye; a hibás állapot olyan (belső) állapota a rendszernek, mely hibahatáshoz vezethet; a hibaok pedig olyan esemény, amely a hibás állapotot kiválthatja. Ezen hibaokok a rendszer határai szempontjából lehetnek külső hibaokok (external faults), de belső hibaokok (internal faults) is.

A szolgáltatásbiztos működést igen sokféle hibaok veszélyezteti. Néhány fontos dimenziójuk: bekövetkezhetnek már a tervezés, vagy csak a működés közben; érinthetik a szoftvert, a hardvert vagy a rendszerkonfigurációt; lehetnek szándékosak vagy szándékolatlanok, jóindulatúak vagy rosszindulatúak; és lehetnek állandósult hibák (permanent faults) vagy átmeneti hibák (transient faults).

A hibahatásokat minőségi jellemzőik alapján szokás hibamód-kategóriákba (failure modes) szervezni. Egy általános, kanonikus megközelítés a hibahatások jellemzésére azok érték- és idődimenzió szerinti, a detektálhatóságot figyelembe vevő osztályozása (Bondavalli–Simoncini, 1990).

Eszerint egy hibahatás az értéktartományban lehet helyes (correctly-valued), kihagyás (omission), durva hibaérték (coarse incorrect value) vagy nem durva hibaérték (subtle incorrect value). A kihagyás esetén a rendszer nem ad szolgáltatást (és ez a

tény önmagában detektálható); a durva hibaérték esetén tartalmában hibás szolgáltatást nyújt, de ez a specifikáció ismeretében könnyen detektálható; míg a nem durva hibaérték kategóriára ez utóbbi nem teljesül. Egy hétköznapi példát, az időjárás-jelentést segítségül hívva: a kihagyás esetén a szolgáltatás egyáltalán nem közöl hőmérsékletértéket, a durva hibaérték esetén a Földön vagy az adott területen fizikailag gyakorlatilag lehetetlen értéket közöl, míg nem durva hiba esetén elméletileg lehetséges, de helytelen – például a valóságtól 20 Celsius-fokkal eltérő – értéket.

Az időtartományban – némi egyszerűsítéssel – megkülönböztetünk helyes időzítést (correctly timed), korai időzítést (early) és késést (late). Bár elfogadhatatlanul lassú informatikai szolgáltatással már minden bizonnyal találkozott minden olvasónk, az nem feltétlenül egyértelmű, hogy miért lehet probléma, ha egy informatikai szolgáltatás túl korán végzi a feladatát (a specifikációban foglalt elvárt működéshez viszonyítva). A fizikai vezérlési feladatok e kategóriának mindenképp fontos példáját adják; egy túl korán fékezésbe kezdő jármű nem a szándékolt helyen fog megállni, míg egy túl korán induló metrószerelvény akár emberéletet is veszélyeztethet. Klasszikus IKT-szolgáltatásoknál pedig példa adatközlés esetén a legkorábbi nyilvánossá tétel időpontjának megsértése (lásd például szavazási eredmények, felvételi ponthatárok stb.).

Ezt az egyszerű, általános és detektálásorientált hibahatás-modellt a gyakorlatban jellemzően tovább finomítjuk (például az adathibák további, a szakterület által megkövetelt taxonomizálásával). Emellett ismert néhány általános érvényű bővítése (lásd például (Gallini–Punnekkat 2011)). Bemutatásának fő szerepe itt annak érzékeltetése, hogy bár egy rendszer rengetegféle módon el tud térni a specifikáció szerinti működéstől, mégis a hibahatások egy alkalmasan választott absztrakciós szinten kifejezően, hatékonyan és minden lehetőséget lefedve osztályozhatók minőségi értelemben.

2.2. Rendszerszintű hibaterjedés

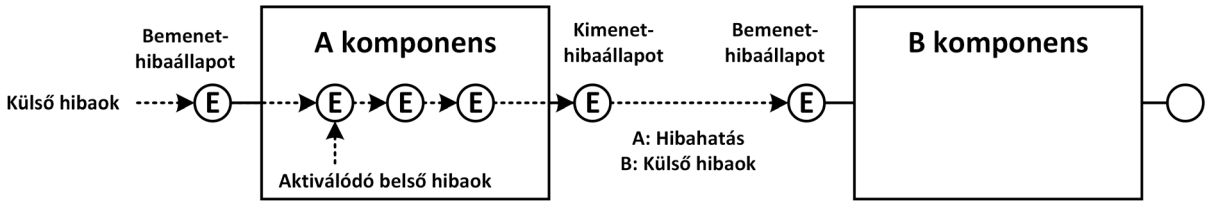
Komplex rendszerekben emellett számolnunk kell azzal, hogy a rendszer valamely részében fellépő hibaokból kialakuló hibás állapot nemcsak hogy „dominószerűen” tovább terjed a rendszerben, a rendszer által nyújtott szolgáltatásban valamely hibahatást okozva; de egyben terjedés közben átalakul, transzformálódik is, mind jellegében, mind pedig súlyosságában. Továbbra is időjárási – bár szándékoltan hipotetikus – példánál maradva: a kritikus téli időszakokban akár néhány fok eltérés is okozhatja egy nem eléggé robosztus intelligens városüzemeltetési rendszerben a síkosságmentesítési és megelőző munkálatok helytelen ütemezését, ezzel akár emberi életre is veszélyes helyzeteket előidézve.

A megoldások rendszerszintű modellezése során az informatika gyakran alkalmaz komponensalapú megközelítést: egy rendszert komponensek hálózataként fogva fel, ahol az egyes komponensek szolgáltatásait, mint „kimeneteket” (output) más komponensek „bemenetként” (input) alkalmaznak. Komponensalapú szemléletben a hibaterjedés (error propagation) a következő folyamatot jelenti.

1. A rendszer egy komponensének belső vagy külső hibaoka abban hibás állapotot indukál, mely a komponens kimenetén mint hibahatás jelentkezik.
2. A komponens kimenetétől – akár vezérlésátadási, akár adatszolgáltatási – értelemben közvetlenül függő további komponensek viszonylatában ezek a hibahatások mint külső hibaokok manifesztálódnak.
3. A külső hibaok ezen komponensekben ismét hibás állapotot indukálhat, potenciálisan hibahatásokhoz vezetve.

4. A hibaterjedési lánc (error propagation chain), ha elér a komponált rendszer határáig, rendszerszintű hibahatást eredményez.

Fontos kiemelnünk, hogy egy adott rendszerben egy adott hibaokhoz, illetve hibaok-kategóriához nem feltétlenül egy hibaterjedési lánc tartozik. A komponensekből összeálló hálózatokban jellemzően különféle elágazások és visszacsatolások vannak jelen, aminek köszönhetően a hiba több úton is terjedhet térben és időben: „elágazik" (párhuzamosan vagy egy ágat választva) és „visszacsatolásra kerül", változatos időzési viszonyok mellett.



1. ábra. Hibaterjedési lánc; a hibaterjedés folyamata
Forrás: a szerzők saját szerkesztése

2.3. Rendszerszintű hibaterjedés korlátozása

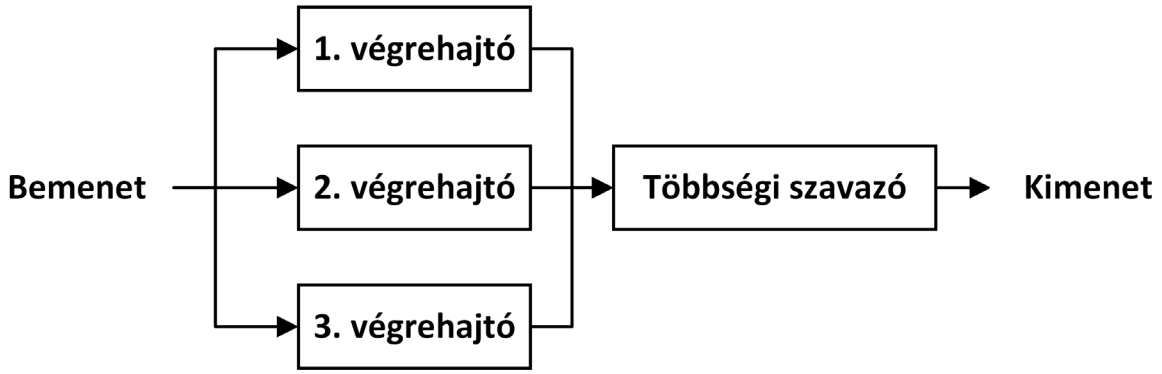
Nyilvánvaló, hogy egy komponált rendszer méretének és komplexitásának növekedésével általánosságban annak az esélye is nő, hogy abban hibák legyenek (például rosszul implementált szoftver), vagy hibák keletkezzenek (például hardverelemek spontán elromlása). A rendszer-kompozíció azonban nemcsak veszélyt rejt magában, de lehetőséget is hordoz a szolgáltatásbiztonsági megfelelőség biztosítása szempontjából; megfelelő tervezéssel és célzott védelmeket megvalósító komponensek beépítésével rendszerszinten biztosítható a hibatűrő működés (fault tolerant operation, fault tolerance), a hibaterjedést tervezett hibabehatárolási tartományokra (fault containment region) korlátozva. A teljesség érdekében itt megjegyzendő, hogy a szolgáltatásbiztonság-biztosítás tevékenységei közül csak az egyik a hibatűrő működés kialakítása; ide tartozik még a hibaokok megelőzése (fault prevention), eltávolítása (fault removal) és előrejelzése (fault forecasting) is, mely fogalmak részben tervezési idejű tevékenységeket fednek.

A hibatűrő számítástechnika, mint önálló diszciplína több évtizede a futásidőben alkalmazható mechanizmusok széles palettáját hozta létre. Ezeknek egy minta-szemléletű, rendszerezett áttekintését adja meg például Hanmer 2013. Általánosságban elmondható, hogy a hibatűrő működés vagy detektálja a hibás állapotokat, és helyreállítást végez (detection and recovery), vagy elfedi hatásukat, és helyreállítást végez (masking and recovery). Megvalósításához mindig valamilyen értelemben vett redundancia bevezetésére van szükség.

Adatredundancia alatt értjük például a különböző hibatűrő kódolásokat és adataudit-technikákat. Időbeli redundanciát használ ki például a szoftvermegfiatalítás (software rejuvenation) (Hanmer, 2010), amely során egyes szoftverkomponenseket időközönként újraindítunk, elkerülendő a belső, illetve tárolt hibák felgyülemlését. Erőforrásbeli redundanciát használnak ki a különböző dedikált ellenőrző- és szavazókomponensek, mint például a háromszoros moduláris redundancia (triple modular redundancy, TMR) (Lyons–Vanderkulk 1962), amelyet a hibatűrési mechanizmusok kiragadott példával érzékeltetésére be is mutatunk.

A TMR-sémában ugyanazt a feladatot három végrehajtó egység végzi, eredményeik egyezőségét pedig egy dedikált szavazóblokk ellenőrzi. Ez a blokk akkor adja tovább

az eredményt, ha legalább két bemenete egyezik, ellenkező esetben hibát jelez. Látható, hogy ez a séma a végrehajtó egységek „minden” hibáját tűri, ha feltételezhetjük, hogy a rendszer teljes működési ideje alatt eléggé valószínűtlen, hogy legalább két egység egyszerre legyen hibás. (Bár a hardver tekintetében sokszor realisztikus, ez a feltételezés azonban kizárja a szoftver hibás voltának lehetőségét is, ha mind a három végrehajtó egység ugyanazt a szoftvermegvalósítást futtatja.) Maga a szavazóegység a legegyszerűbb esetben természetesen egyszeres hibapont (single point of failure), de egyszerűsége és kiemelt fontossága miatt meghibásodási valószínűsége jellemzően nagyságrendekkel kisebb, mint a szavaztatott egységeké (emellett extrém küldetéskritikus alkalmazásokban, mint például az űrszondák, alkalmaznak TMR-szavazókat TMR-sémában elrendező megoldásokat).



2. ábra. A TMR-séma
Forrás: a szerzők saját szerkesztése

A hibatűrő számítástechnika eszközkészletét az általános célú számítástechnika a redundanciák „ára” miatt sokáig csak korlátozottan használta; a TMR például kifejezetten az „ultrakritikus” alkalmazásokra volt jellemző. Az okos város és okos alkalmazások szempontjából viszont kifejezetten releváns, hogy felhőplatformokon a TMR és hasonló sémák alkalmazása ma már kommerszebb feladatokra is reális lehetőség; egyrészt a megváltozott kockázati és költségprofilok, másrészt a gyors és könnyű újrakonfigurációs lehetőségek miatt (bővebben lásd Bozóki–Koronka–Pataricza 2015).

3. MODERN HIBATERJEDÉS-ELEMZÉS

A hibaterjedés-elemzés (változóan error propagation analysis, illetve failure propagation analysis, a továbbiakban EPA) a lehetséges hibaterjedések rendszerszintű elemzése és a „hiba nem terjedések", azaz a hibabehatárolási tartományok legtöbbször pesszimista becslése kvalitatív (azaz nem valószínűségi numerikus) hangsúllyal.

Klasszikusan ezt a tevékenységet a rendszertervezés, majd verifikáció és validáció (V&V) során manuálisan végezték; valójában az olyan szabványos technikák, mint az FTA vagy az FMEA is egyfajta EPA-t valósítanak meg. Ezen módszerek egy áttekintését adja az első fejezet.

A kritikus alkalmazások modellvezérelt fejlesztésének (Model Driven Engineering) elterjedésével (Whittle et al. 2014) azonban megnyílt a lehetőség azon erősen automatizált módszerek irányába, amelyek a fejlesztés során létrejövő követelmény-, tervezési és megvalósítási modelleken hajtják végre az elemzést. A formálisan megkövetelt, adott esetben ellenőr, illetve felügyeleti szerv által is ellenőrzött FTA/FMEA/FMECA/CCA (Pietrantuono–Russo 2013) modellek ma már ezen módszerek automatikusan generált termékei (modern fejlesztési környezetet és metodológiát feltételezve).

Ezt a mintát valósítják meg többek között az xSAP (Bittner et al. 2016), Compass Toolset (Bozzano et al. 2011) és HiP-HOPS (Wolforth et al. 2010) eszközök. Kihívást jelent ma még a rendszer-szoftverkomponensek hibaterjedési tulajdonságainak meghatározása, különösen a tervezés korai fázisaiban; a közvetlenül végrehajtható specifikációk megjelenésével azonban e területen is jelentős előrelépések tapasztalhatók, lásd például Bonfiglio et al. 2015.

Általánosítva, de mondhatjuk azt, hogy az FTA és FMEA jellegű módszerek (illetve elemzési termékek) létrehozásának fő célja a fejlesztési folyamatban elhelyezve azokat egyrészről az, hogy a hazárdhoz vezető komponens-hibahatásokat azonosítsák. Másrészről a követelmények szempontjából elfogadhatatlanul magas kockázatú hazárdok – például futásidejű vagy strukturális megoldások segítségével – elégséges kezelésének tényét demonstrálják.

Ezen módszerek okkal adják évtizedek óta a szolgáltatásbiztonságra tervezés elemzési eszközkészletének egyik alapját. Használatuk jellemzője azonban, hogy a tervezés egyes fázisaiban tipikusan döntésreaktívak; a már kialakított és védelmekkel ellátott funkcionális architektúrán, rendszerarchitektúra-terven stb. kerül meghatározásuk elvégzésre. Míg ez a megközelítés egy sor rendszertípus esetén alkalmas választás – a gyártási folyamatoktól az űrrakéták tervezéséig –, a tapasztalat és a jelenleg is futó kutatások azt támasztják alá, hogy a kifejezetten szoftverfókuszú, illetve dinamikus szolgáltatásintegrációt megvalósító rendszerekre nem az. (A tárgyszerűség érdekében kijelentésünket annyival szükséges ellenpontoznunk, hogy kifejezetten durva felbontású redundancia esetén az FMEA alkalmas a mitigációs mechanizmusok kiválasztására is.)

Ezekben a rendszerekben – ismét csak erősen egyszerűsítve – tipikusan nem az a fő kérdés, hogy „milyen típusú és megbízhatóságú tranzisztor legyen beépítve", hanem hogy az alkalmazható nagyszámú hibatűrési minta közül mi az a készlet, amely a felmerülő sokféle igényt – üzleti igényektől a biztonságossági kritériumokig – mind kielégíti. Ehhez a tervezési modellek kialakításának integráns részévé érdemes tenni az EPA

tevékenységeket, lehetőség szerint intelligens algoritmusokkal támogatva a hibatűrés mechanizmusainak kiválasztását. Másrészről pedig kezelendő az a tény, hogy a szoftverrendszerek (figyelembe veendő) hibaokai, hibás állapotai és hibahatásai különösen sokfélék lehetnek; kiváltképp a tervezés korai fázisában, amikor is a funkcionális viselkedés még legalább részben csak részlegesen specifikált.

Így a modern, modellalapú fejlesztési folyamatokba integrálódó EPA modellezési és következtetési logikája is más, mint az eredetileg manuálisan végrehajtott induktív/deduktív következtetési megközelítések esetén.

3.1. Kvalitatív hibaautomaták

A modern, általánosított EPA egy, a matematikai absztrakcióelmélet szempontjából formálisan is megalapozott megközelítését adja meg Pataricza 2008a; rövidített áttekintést ad Pataricza, 2008b. A metodológia egyik alapeleme, hogy a rendszerkomponensek (hibamentes és hibás esetekben is megadott) funkcionális specifikációjából és a komponensek specifikációtól való eltéréseinek kategorizálásából (amelynek egy példája a fent hivatkozott, Bondavalli–Simoncini-féle taxonómia, s amely valójában egy, az eltérések feletti kvalitatív absztrakciót fogalmaz meg) algoritmikusan komponens-hibaautomatákat képez. Ezek koncepcionálisan nagyban hasonlítanak a ma már például az AADL által deskriptív szinten támogatott hibaautomatákhoz, mint az később a 4.4.2 pontban bemutatásra is kerül. (Az algoritmika jellegzetességeinek és alapötleteinek bemutatására itt nem vállalkozunk, mivel az kifejezetten számítástudományi alapismeretek feltételezését igényelné.)

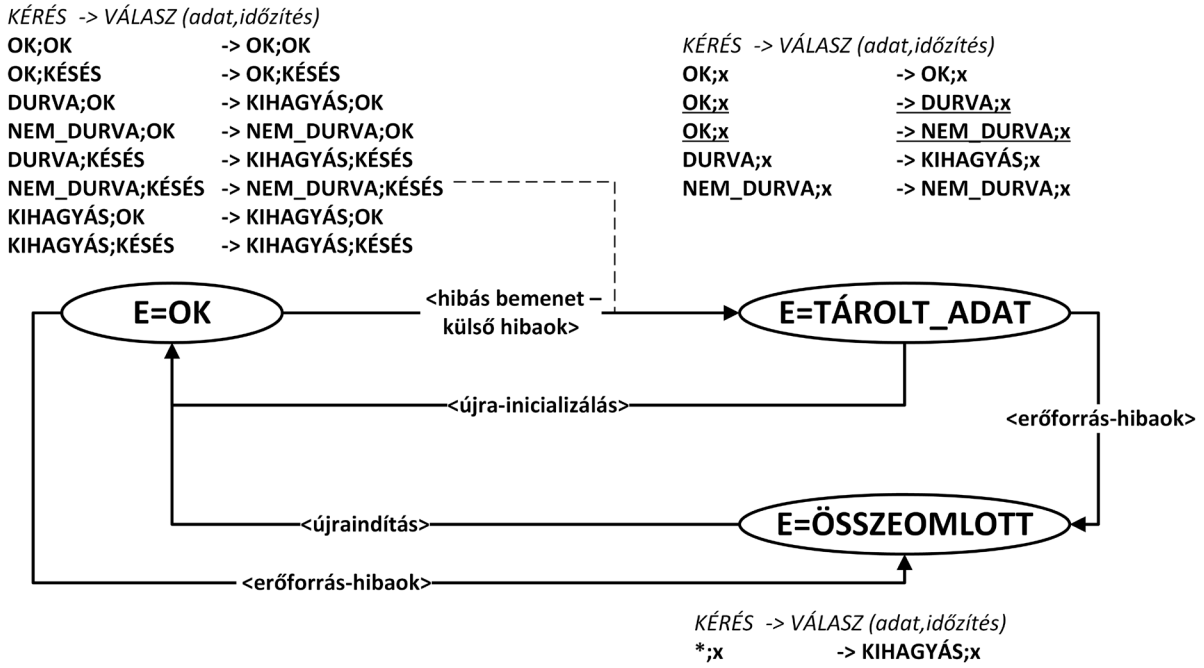
Ezek a hibaautomaták és a hibaautomatákból képzett automatahálózatok az informatika formális módszereinek segítségével már vizsgálhatók (illetve vizsgálatukból és szimulációjukból FMEA, FTA stb. kimenet készíthető). Megjegyzendő, hogy míg Pataricza 2008a általános értelemben lefekteti a hibaautomaták (végrehajtható) specifikációból előállításának keretrendszerét, a hibaautomaták létrehozásának nem ez az egyedüli módja; azokat felállíthatjuk szakértői tudás segítségével, de kikövetkeztethetjük rendszermegfigyelésekből is. Az utóbbit lehetővé tevő „log mining” algoritmikája és gyakorlata jelentős; egy kiragadott, szemléltető példaként referáljuk Jiang és szerzőtársai 2007-es munkáját (Jiang et al. 2007), mely a „helyes” és „helytelen” automataviselkedések mintaalapú, általánosító tanulására ad megközelítést. Specifikusan a kvalitatív hibaautomaták tanulásának kontextusában meg kell említenünk, hogy a komponens-hibaérzékenységi és -hibaterjedési modellek kísérletekből adatbányaszati eszközökkel kinyerésére is ismertek kidolgozott megközelítések (lásd például Pintér et al. 2008 és Pintér et al. 2005). A szakértői tudás alapján modellezés alapvető elemeit bemutatja például Ge-Paige–McDermid 2009 és Csertán–Pataricza et al. 2002.

3.2. Statikusszindróma-alapú hibaterjesztés

A hibaterjedési hálózatok dinamikus viselkedésvizsgálata helyett azonban lehetőségünkben áll „könnyű fajsúlyú” formális módszerek alkalmazására is („lightweight formal methods”), méghozzá egy úgynevezett időbeli absztrakció elvégzésével.

Példaként tekintsük egy kérés-válasz jellegű szolgáltatás – ilyenek például a webkiszolgálók – kvalitatív hibaautomatáját. Megjegyezzük, hogy a grafikus reprezentáció alapvetően szemléltető célú; formálisan (illetve számítógépes feldolgozásra alkalmas formában) az automatát állapotvektorokkal és az állapot-vektorok felett értelmezett ún. állapot-átmeneti relációval adnánk meg.

A modellezett rendszerkomponensnek három fő hibaállapota van: helyes működés; tárolt adathiba; és összeomlott állapot. A tárolt adathiba különösen fontos – és sokszor nehezen kezelhető – az informatikában, mivel a hibaok jelentkezése után még sokáig tud lappangani (az adathozzáférés mintázataitól függően), csak jóval később okozva igen komoly hibahatásokat. Leghétköznapibb példája, amikor egy ügykezelési folyamat kezdetén az ügyfél valamely adata – például életkor – rosszul kerül egy adatbázisban rögzítésre, és a munkafolyamat további lépései ezt a helytelen adatot használják a továbbiakban bemenetként.



3. ábra. Egyszerű kvalitatív hibaautomata
Forrás: a szerzők saját szerkesztése

Az egyes főállapotokhoz rendelt táblázatok azt adják meg, hogy a kvalitatív hibamódokkal jellemzett kérésekre milyen kvalitatív hibamódokkal jellemezhető válaszokat adhat a kiszolgáló. A feltételes mód fontos; míg a hibamentes állapotban determinisztikusak a kérésekre adott válaszok hibamód-leképezései, illetve transzformációi, addig a tárolt hiba esetén nem determinisztikus eseteket is megengedünk; azt fejezve ki, hogy tárolt hiba esetén helyes adattartalmú kérésre is adhat potenciálisan helytelen választ a kiszolgáló. Míg a hibamentes állapothoz rendelt táblázat – didaktikai céllal – minden kombinációt kiír, addig a másik kettő él egyszerű mintákkal; a tárolt hiba esetén a bemenet „időzítési" kategóriáját „átmásolva" a kimenetbe, az összeomlott esetben pedig a bemenet adathelyességét „érdektelenként" (csillag karakter) jelölve. Ehhez hasonló, egyszerű hibatranszformáció-leíró nyelvek ismertek; két meghatározó példa a Generalized Failure Logic (GFL) (Wolforth et al. 2010) és a Fault Propagation and Transformation Calculus (FPTC) (Wallace 2005).

Ezen automata felett – különösen, ha a főállapot-átmenetek, illetve a nem determinisztikus válasz-választások valószínűségi jellemzőit is megadjuk – már végezhető nemcsak klasszikus induktív és deduktív EPA, de stochasztikus módszerekkel a megbízhatóság és a rendelkezésre állás számszerű vizsgálata is. A hibaautomaták természetesen rendszerré komponálhatók, a rendszertopológia által meghatározott módon bemeneteket kimeneteknek feleltetve meg.

Példaautomatánknak azonban van egy jóval egyszerűbb nézete is. Nemcsak az egyes bemeneti és kimeneti hibákat, hanem egy rendszerfutás (egy időintervallumon folyamatos működtetés) során fellépő bemeneti és kimeneti hibasorozatokat is osztályozhatjuk, az alapján például, hogy mi az általuk tartalmazott „legsúlyosabb" hiba. (Kifejezetten az elméleti érdeklődésű olvasóknak megjegyezzük, hogy ennek egy lehetséges, matematikailag precíz módja a potenciálisan végtelen futások végtelen halmazának temporális logikai formulákkal particionálása.) A hibák súlyossági besorolását sokszor szakterületi szabványok segítik; szoftverrendszerekre ilyen például az IEEE 1044-2009 – Standard Classification for Software Anomalies.

Így az összetartozó bemenet-kimenet sorozatok helyett egy bemenetsorozat-hibaosztály – kimenetsorozat-hibaosztály páros fog leírni egy-egy automatafutást (egy bemenetet és egy kimenetet feltételezve). Ezek a megfelelések már egyszerű táblázatos formában – matematikailag ún. relációként – is felírhatók. A relációsorok elemeit szokásosan szindrómáknak nevezzük (Pataricza 2008).

Erre mutat példát az alábbi táblázat. Látható, hogy az aluldeterminháltságot az „időbeli absztrakció" csak növeli; ez bizonyos értelemben elkerülhetetlen ára a kompaktálásnak. Kissé furcsán hathat az S5, „minden egyéb" szerepű szindróma is. Ennek alkalmazását a példa szándékolt egyszerűsége is indokolja, de az is, hogy a hibaterjedés-vizsgálatban legtöbbször megkötjük a vizsgált rendszer hibaaktivációit, például legfeljebb egyetlen aktív belső hibaokot feltételezve (single fault assumption). Ennek hatására egy olyan rendszer elemzése során, ahol már korlátozzák védelmi mechanizmusok a hibaterjedést, a kevert hibamódok megjelenése atipikus.

Az ilyen, ún. statikus szindrómarelációk kvalitatív hibaautomatákból meghatározásának precíz algoritmikáját megadja Pataricza 2008; de itt is adott a lehetőség, hogy a relációt mérnöki tudás alapján határozzuk meg.

Belső hibaok	**Bemeneti futás (adatdimenzió)**	**Kimeneti futás (adatdimenzió)**
Nincs	csak OK **(S1)**	csak OK **(S1)**
Nincs	van KIHAGYÁS, de egyébként legfeljebb OK **(S2)**	van KIHAGYÁS, de egyébként legfeljebb OK **(S2)**
Nincs	van DURVA, de egyébként legfeljebb OK **(S3)**	van KIHAGYÁS, de egyébként legfeljebb OK **(S2)**
Nincs	van NEM_DURVA, de egyébként legfeljebb OK **(S4)**	csak OK **(S1)**
Nincs	van NEM_DURVA, de egyébként legfeljebb OK **(S4)**	van DURVA, de egyébként legfeljebb OK **(S3)**
Nincs	van NEM_DURVA, de egyébként legfeljebb OK **(S4)**	van NEM_DURVA, de egyébként legfeljebb OK **(S4)**
Nincs	S1, S2, S3 és S4 által nem fedett esetek (kevert hibamód) **(S5)**	**S1** VAGY **S2** VAGY **S3** VAGY **S4** VAGY **S5**
Összeomlás	**S1** VAGY **S2** VAGY **S3** VAGY **S4** VAGY **S5**	van KIHAGYÁS, de egyébként legfeljebb OK **(S2)**

3. táblázat. Egyszerű kvalitatív automatából képzett statikus hibaterjedési reláció
Forrás: a szerzők saját szerkesztése

A statikus szindróma reláció alapú hibaterjedés-modellezés egyértelmű előnyét a következők adják.

1. A rendszer-kompozíciót bemeneti és kimeneti szindrómaazonossággal modellezve az elemzési feladatok a táblázatokon végigvezetett, azok által közösen megengedett, komponensenkénti bemenet-kimenet-belső hibaok kombinációk keresésévé egyszerűsödnek.
2. Az ilyen jellegű keresések célszoftverekkel jól támogatottak (például az ún. korlátkielégítés-programozás, Constraint Satisfaction Programming – CSP – területén: Szeredi–Benkő, 2002), de a legegyszerűbb esetekben manuálisan is végigkövethetők. A matematikai keresési feladatra leképezés ismertetésétől itt – jelen munka jellegéből adódóan – eltekintünk.
3. Az alkalmazott (vagy nem alkalmazott) védelmek is modellezhetők ebben a szemléletben, gyakorlatilag azt adva meg, hogy alkalmazásuk mely terjedési kombinációkat nem engedi meg. A fenti kiszolgálónál például egy kimeneti hihetőségvizsgálatot végző megoldás a NINCS:S4->S3 kombinációt könnyedén eliminálni tudná. A matematikai kombinációkeresés részévé lehet tenni azt is, hogy mely védelmek „bekapcsolása” szükséges a kívánt hibabehatárolási tulajdonságokhoz.
4. A modellezési és elemzési módszer kiemelkedő előnye, hogy a „keresés irányát” – azaz hogy induktív vagy deduktív EPA-t végzünk – egyáltalán nem határozza meg előre. Adott kimenetekhez kereshetünk lehetséges belső hibaokokat és bemeneteket (root cause analysis); adott bemenetekhez és belső hibaokokhoz kereshetünk lehetséges kimeneteket (hatásvizsgálat); adott bemenetekhez és belső hibaokokhoz kereshetjük a védelmeket, amelyek adott kimeneteket eliminálnak (mechanizmustervezés); és ezen tevékenységeket végezhetjük kevert módon is.
5. Bár a modellezés stílusából adódóan egy-egy elemzési feladat esetén több, alternatív hibaterjedésilánc-hipotézis is adódhat megoldásként (ez elemi, komponensszinten már a példarelációban is látható), a hipotézishalmazok minőségi jellemzése is lehetséges, ahogy azt például Kocsis 2018 bemutatja. Így az elemzések megfogalmazhatók úgy, hogy például „meddig terjednek a rendszerben az S4 és S5 hibahatások, ha egyszeres belső hibaokot és S1-S2-S3 bemeneteket feltételezünk”.

3.3. EPA az okosváros-módszertanokban

A (statikus szindróma reláció alapú) EPA alkalmazható klasszikus fejlesztési folyamatokba integrálva, a hazárd- és kockázatkezelés részeként. Emellett azonban az EPA megteremti annak lehetőségét is, hogy gyorsan és hatékonyan – bár az absztrakcióból és a tudáshiányból adódóan pesszimistán – tudjunk érvelni hibabehatárolási tartományokról magas szintű, a hibaterjedést felderítő és a védelmi stratégiákkal gondolatkísérleteket végző vizsgálatok során. Ez a képesség várhatóan a kiberfizikai rendszerek, a kiberfizikai rendszer jellegű okosváros-szolgáltatás-szövetek és a természeti katasztrófáknak, fizikai támadásoknak ellenálló városok (resilient cities) (Godschalk 2003) területein fog a jövőben jelentőséget nyerni.

Az informatika egyik napjainkban is zajló forradalma a „System of Systems” („rendszerek rendszere”, SoS) jellegű kiberfizikai rendszerek (Rajkumar et al. 2010) térnyerése. Kiberfizikai rendszer (Cyber-Physical System, CPS) alatt értjük azokat a számítógépes rendszereket, amelyek a fizikai világot megfigyelő és abba intelligens módon beavatkozó,

hálózatba kötött, kooperáló „terepi” elemekkel rendelkeznek. A CPS intelligenciájának fontos eleme, hogy a komoly kapacitásokat igénylő adattárolási és -feldolgozási feladatokra távoli, illetve a „a terep és az internet határán” elhelyezett (edge) felhőszolgáltatásokat használnak.

Az okos városok legtöbb definíciója és értelmezése a holisztikusan és sokszor dinamikusan – integrált IKT-megoldásokat az „okosabbá tétel” („smartness”) kulcselemeként értelmezi (lásd például Garcia et al. 2015, mind az olyan – Neirotti et al. 2014 terminológiájával élve – „kemény” területeken, mint az energiahálózatok, közbiztonság, közlekedés, szállítmányozás és egészségügy, mind pedig az olyan „puha” területeken, mint az oktatás, kultúra és közigazgatás. Ezen megoldások nagy része azonban egyben kiberfizikai rendszer is, és kompozíciójuk, az okos város is az. (Valójában az egyes okosváros-alrendszerek hierarchikusan, CPS-ek SoS-eiként értelmezhetőek, de ezen aspektus részletes tárgyalásától itt eltekintünk.)

Mint korábban már rámutattunk, általánosságban az IKT-megoldások – és specifikusan a CPS-ek – a városi élet szövetébe beépülése komoly kockázatokat is rejt magában a merőben újszerű lehetőségek mellett.

A városi rendszerekben ugyanúgy szükséges vagy szükséges lenne a kockázatok azonosítása, kiértékelése, elemzése, kezelése és monitorozása, mint bármely „elégségesen fontos” üzleti/ipari folyamatban. Ennek metodológiai támogatása legalább a kritikus műszaki alrendszerek és a természeti és egyéb katasztrófák esetén kialakult és alkalmazott.

Az okos városok azonban az okosváros-funkciók és informatikai rendszerek integrálásával új aspektusként szükségszerűvé teszik az alrendszerek közötti hibaterjedésre vonatkozó tulajdonságok tervezését és kezelését. Ehhez az EPA megfelelő eszközkészletet nyújt, de önmagában csak egy hatékony elemzési eszköz – holisztikus, kifejezetten okos város kockázatkezelési metodológiát a szerzők nem ismernek. Közérthetően fogalmazva: az okos városok alaptulajdonságának tekinthető rendszer- és infrastruktúra-kombináció, összekötés és integrálás egyben a szeparáltság nyújtotta védelmet is megszünteti; így a város mint rendszer szintjén el kell kezdenünk foglalkozni a rendszerek közötti hibaterjedés kockázatvezérelt kontrollálásával.

4. HIBATERJEDÉS-ELEMZÉSI MÓDSZEREK ALKALMAZÁSA OKOSVÁROS-KÖRNYEZETBEN

A hibaterjedés elemzésének különböző módszerei okosváros-környezetben különösen indokoltak, elsősorban a széttagolt, egymással kapcsolatban álló infrastruktúraelemek, ezek funkcionalitása és az ezekre épülő szolgáltatások miatt. A hibaterjedési módszerek alkalmazásának előfeltétele az okos városi környezet és szolgáltatásmodell rendelkezésre állása. A modell definiálása történhet strukturált szöveg/táblázat formájában, de ennél hatékonyabb, egyértelműbb és karbantarthatóbb, ha a modell definiálása egy erre kifejlesztett modellezési nyelven történik. Ennek bővebb leírását lásd a Kritikus infrastruktúrák és szolgáltatások modellezése kismonográfiában (Gönczy 2018).

A hibaterjedés-elemzés célja okos városi környezetben többes lehet:

- Egy adott komponens vagy szolgáltatás megfelelőségének vizsgálata (a rendszerbe helyezve gyengíti-e a rendszer megbízhatósági tulajdonságait, megfelel-e az elvárásoknak). Erre példa lehet annak vizsgálata, hogy egy külső szolgáltató által üzemeltetett szolgáltatás okozhat-e olyan hibát, amely a teljes rendszer téves működéséhez vezethet. Példa: a BKK interaktív térképe használja a Waze rendszerből származó információkat, amelyek azonban a felhasználói bejelentéseken alapulnak. Kérdés, a Waze adatainak hibája befolyásolhat-e olyan információt, amely a program felhasználói körén kívül jelentős tömegeket érinthet, így lehetséges-e kisszámú, „rosszindulatú” téves bejelentéssel zavart okozni.
- Egy adott feladat vagy komponens esetében mik az elvárt/lehetséges/megengedett hibamódok. Például egy adott távolságmérő szenzor hibája lehet elhanyagolható, ha a szenzor forgalmi statisztikák gyűjtéséhez szolgáltat adatot, és lehet súlyos, ha a később bemutatásra kerülő intelligens kereszteződés esetében a szenzor a járművek kereszteződéstől való távolságát méri. Hasonlóképpen, ha egy adott terület foglaltságát mérő alkalmazás egy intelligens parkolóházban fut, akkor kisebb jelentőségű a hibája, mintha például egy vészkijárat foglaltságáról kellene riasztást generálni a jelzése alapján. Szintén eltérhet, hogy egy adott alkalmazás szempontjából az elnyelés (omission) vagy a téves adattovábbítás számít-e súlyosabb hibának, illetve melyiknek mi a hatása az alkalmazás működése szempontjából.

4.1. Kaszkád hibahatások

Kritikus infrastruktúrák esetében kiemelt fontosságú azon szcenáriók vizsgálata, melyek kaszkád hibahatások eredményeként alakulhatnak ki. Például egy természeti katasztrófa esetében az úthálózat, az elektromos hálózat és a telekommunikációs rendszerek meghibásodásának hatásai „összeadódhatnak” ezek együttes jelentkezése esetén. Ezt nevezi a köznyelv „dominóhatásnak”, de a szakirodalom is foglalkozik ennek elemzésé-

vel. Egyes szakterületeken belül, például elektromos átviteli rendszerekben ennek önálló szakirodalma és gyakorlata van (például Wang, Dobson, Tan), amelyek az elektromos hálózatban és a termelésben bekövetkező zavarok lehetséges terjedését vizsgálják, a kár behatárolásával, illetve a kárbehatárolási régió (Damage Confimenement Region) lehetséges leszűkítésével foglalkoznak. Jelen tanulmányban olyan példákra koncentrálunk, amelyek a több, összekapcsolt infrastruktúra működésének hatásait vizsgálják.

Ennek vizsgálatára több módszert létezik, többek közt a „kaszkád hibák elemzése" (Cascade Effect Analysis), mely metodikáját tekintve tekinthető az EPA-fejezetben bemutatott módszer egy lehetséges alkalmazásának. Hasonlóan fontos a korrelált hibák hatásának vizsgálata, amely a szisztematikus hibaterjedési modellekkel szintén lefedhető.

4.2. Kritikus infrastruktúrák folyamatainak ellenőrzése szimulációval

A 3. fejezet bemutatta, hogyan lehet formális modellek használatával hibaterjedés elemzési vizsgálatokat végrehajtani. Ennek egyik alternatívája lehet a szimulációs vizsgálat.

Dinamikus modellek ellenőrzésének egyik lehetősége a diszkrét szimulációs módszerek használata. Diszkrét szimuláció esetén a rendszer viselkedését úgy közelítjük, hogy elkészítjük a rendszer szimulációs modelljét. Ez a modell a rendszer konkrét viselkedését valamilyen értelemben közelíti, a viselkedési logika főbb elemeit követi, ám nem igényli a rendszer konkrét megvalósítását. A szimulációs modell sok esetben a rendszer tervezésekor elkészülő mérnöki modell valamilyen kiegészítése.

A szimuláció működéséhez meg kell határozni a rendszer bemeneteit, ami üzleti folyamat modellel leírt rendszerek esetében (ahogyan azt a Gönczy 2018 monográfia is bemutatta).

Követve a kvalitatív absztrakció elvét, a rendszer bizonyos paramétereit megkötve a rendszer bemenetére eseményeket, ún. színezett tokeneket adunk (normál tartomány – gyanús érték – hibás érték, ennek finomítása lehet a hibatípusok megkülönböztetése). Erre több elérhető üzleti folyamat modellező eszköz ad támogatást.

A szimuláció ezek után kétféle módon veheti figyelembe a hibaterjedés és -kezelés mechanizmusait:

- Amennyiben az adott eszköz erre alkalmas, a szimuláció nemcsak a véletlen sorsolás alapján választ az utak közül, hanem érték szerint. Tehát a kiértékelés során a folyamatmodellbe be kell építeni a hibás terjedést és viselkedést jelző lépéseket, amelyeket az eszköz kiválaszt, ebben az esetben már a bemeneten jelezni kell, mely munkadarabok (pontosabban az adott munkadarabokat feldolgozó folyamatok) lesznek a későbbiekben hibásak.
- Ennél fejlettebb eszközök képesek a „tokenek" változásait is lekövetni a szimulációban, megadható egy bemenet-kimenet leképezés („a hibás űrlap hibás kérvényt eredményez", „a túl kicsi érték helyett a megengedett minimum értéket továbbítjuk").

Lehetőség van arra is, hogy (voltaképpen a folyamat végrehajtását leegyszerűsítve) a folyamat szimulációját teljes egészében ún. üzleti szabály végrehajtó rendszerekben (Business Rule Management System) végezzük el. Ezek a szabályok „HA-AKKOR" struktúrát követve írják le a rendszer egyes részelemeinek viselkedését adott szituációkban.

A folyamat alapú szimuláció értelemszerűen csak „forward propagation” (tovaterjedés) kérdéseket tud megválaszolni, vagyis azt tudja vizsgálni, hogy egy rendszer/komponens bemenetére csatolt változónak mi lehet a hatása.

Ha a hibaszimulációt és hibaterjedési analízist összehasonlítását nézzük elmondható, hogy a folyamatszimulációra épülő vizsgálat voltaképp egy egyszerűsített EPA azzal a kiegészítéssel, hogy alacsonyabb absztrakciós szinten ír le olyan fogalmakat (például hibaállapot), melyek az EPA matematikai formalizmusának részei.

A szimuláció emellett nem tekinthető kimerítő vizsgálatnak, így nincs arra garancia, hogy megtalálja az összes lehetséges lefutást, azaz az összes lehetséges kimenetet egy adott folyamat mellett. Előnye ugyanakkor, hogy nem igényel matematikai apparátust és a legtöbb üzleti folyamat modellező környezetben beépített támogatás van a végrehajtásához. Előnye még, hogy kimenete közvetlen az adott folyamaton értelmezhető. A folyamatszimuláció adott esetben felfogható úgy is, mint a formális elemzési módszerek egyfajta szakértői validálása.

4.3. Kvalitatív hibaterjedési vizsgálatok támogatása mérnöki modellekkel

Az FMEA, EPA vagy ahhoz hasonló megközelítésre épülő hibaterjedés elemzési módszerek közös jellemzője, hogy elvégzésükhöz a rendszer számos aspektusát ismerni kell.

- Követelmények: A hibaterjedés alapelve, hogy legalább kvalitatív szinten meg kell tudni különböztetni a jó, elfogadható vagy már hibás elemeket, legyen szó akár adatról, akár egy szolgáltatás működéséről, akár egy erőforrás állapotáról.
- Funkcionalitás: A modellnek le kell tudnia írnia a rendszerelemek alapvető funkcionalitását.
- Topológia: Mivel a legtöbb hiba terjedéséhez direkt vagy implicit logikai kapcsolat szükséges, a modellnek tartalmaznia kell a modellelemek közti kapcsolatokat is, azok típusával együtt.
- Adatáramlás: Szoftveres hibaterjedési vizsgálatok alapvető elemek az adat és az adatáramlás definiálása. Ehhez a legtöbb módszer esetén valamilyen topológiai kapcsolatnak (vagy kapcsolat láncnak) is fenn kell állnia.
- Erőforrás-hozzárendelés: Okos városi folyamatokban különösen igaz, hogy a szoftvert a kiszolgáló infrastruktúrával érdemes vizsgálni, ideértve akár a „szerver” jellegű számítógépeket, akár a beágyazott eszközöket, szenzorokat, akár a mobil/okos eszközöket. Ezek hibamodellje a szoftverben okozhat olyan váratlan viselkedést (például időzítési, adattovábbítási hibák), amelyre a szoftvert nem feltétlenül készítették fel. Emellett az egyes funkcionalitások végrehajtásának körülményei is erősen függenek a végrehajtó egység képességeitől. Mivel manapság már a végrehajtás egyik tipikus eszköze felhő (ideértve akár egy városi infrastruktúra keretein belül üzemeltetett privát felhőt), így ennek a sajátosságai is befolyásolják a rendszer működését, például adott esetben az egy fizikai erőforráson futó szolgáltatások hatással lehetnek egymásra.
- Fizikai világ jellemzői: Az okos városi szolgáltatások, ahogyan az a kutatóműhely több tématerületében is megvizsgálásra került, erősen függenek a fizikai világból nyert adatoktól. Ezen adatok megbízhatósága, külső tényezőktől (hőmérséklet, páratartalom, háttérzaj, fényviszonyok stb.) való függése és a fizikai paraméterek informatikai reprezentációja (mely egyben felfogható akár egy kezdeti kvalitatív modellre történő leképezésnek) önmagában is vizsgálandó terület. Feltehető például a kérdés, hogy adott fizikai körülmények és berendezések/szenzorok mellett előfordulhat-e, hogy egy adott útszakaszt üresnek érzékelünk, holott fog-

lalt, vagy egy jármű sebességét a fényviszonyok miatt alábecsüljük egy optikai mérőeszközzel, stb. Ehhez hasznos, ha már a mérnöki modellben meg tudjuk jeleníteni a fizikai világ paramétereit.
- Hibák hatását csökkentő, illetve elhárító folyamatok és tényezők: A hibaelhárítás folyamatai tipikusan leírhatók a normál működéssel megegyező módon (például folyamatmodellel, vagy állapotgéppel), vannak azonban olyan modellezési megközelítések (például AADL), ahol ezek kitüntetett formában leírhatók, és hozzákapcsolhatók az adott hiba megjelenési pontjához.

A mérnöki modellek és a hibamodellezés összekapcsolásának előnyei:
- Az egyes problémák már a rendszertervezés korai szakaszaiban azonosíthatók (például egyszeres hibapontok, közös módusú hibák, hiányzó hibakezelés stb.).
- A különböző architekturális minták a tervezés korai szakaszában összehasonlíthatók és kiértékelhetők.
- A rendszermodell fejlődésével a vizsgálatok ismételten végrehajthatók, az eredmények összehasonlíthatók.

Mérnöki modellek megbízhatósági tulajdonságokkal történő kiegészítésére és ezáltal azokból automatizáltan származtatott megbízhatósági modellekre ad példát a Majzik–Pataricza–Bondavalli A. 2003 cikk.

4.4. Modellezési nyelvek

Kiegészítve a Gönczy 2018 tanulmányban leírtakat, áttekintjük a legismertebb modellezési nyelvek támogatását a hibaterjedési vizsgálathoz szükséges modellelemek reprezentációjának lehetősége szempontjából.

4.4.1. Systems Modelling Language (SYSML)

A Systems Modelling Language (SysML) a szoftverrendszerek általános célú modellezésére megalkotott UML (Unified Modelling Language) egy kiterjesztésének tekinthető, amely támogatja egy teljes endszer tervezési folyamtának minden lépését, ideértve a követelmények modellezését és a fizikai világ paramétereinek, időzítési tulajdonságainak megadását.

A SysML nyelv több diagramja is fontos szerepet játszik a hibaterjedési vizsgálatok tipikus elvégzésében (például David–Idasiak–Kartz 2010; MHeni 2013):
- Követelmény diagram (Requirements): A SysML lehetőséget ad arra, hogy a rendszer funkcionalitásával szemben támasztott követelményeket strukturált formában adjuk meg, oly módon, hogy később azok modell elemekhez legyenek rendelhetők.
- Blokk definíciós diagram (Block Definition Diagram): a blokk definíciós diagram adja meg a rendszer alapvető felépítését, és az egyes elemek közti kapcsolódási pontokat, így a hibaterjedési módszerek alapvető bemenetéül szolgál.
- Aktivitás diagram (Activity Diagram). A rendszer dinamikus működését írja le, hivatkozva az egyes lépéseket megvalósító elemekre, illetve metódusokra. A

rendszer topologikus felépítését leíró blokk diagrammal kombinálva, valamint felhasználva a SysML által nyújtott, időzítési és egyéb paramétereket leíró nyelvi elemek lehetőségeit, az aktivitás diagramok megadják az egyes hibák logikai terjedési útjait, az elemek közti adat- és vezérlési függőségeken keresztül.

4.4.2. Architecture Analysis & Design Language (AADL)

Az AADL nyelvet eredetileg repülők tervezésére alkották meg (eredeti neve Avionics Architecture Description Language). A nyelv elemeiről és használatáról jó áttekintést ad (Feiler–Glutch 2012). Az AADL felépítéséből és alapelveiből adódóan alkalmas a hibaterjedés hatásainak vizsgálatára.

Az AADL alapvetően a fizikai világgal kapcsolatban lévő, valós idejű működésű, erős biztonsági követelményeknek megfelelő rendszerek leírására alkalmas. A tervezési folyamatnak szerves része a modellek ellenőrzése (mind funkcionális, mind például időzítési szempontból, utóbbi kiemelten fontos a komponensen kommunikációjának tervezésekor). Az alábbiakban a hibamodellezés helyét és szerepét mutatjuk be (Delange–Feiler 2014) alapján, kifejezetten a nyelv hibamodellje (Error Model Annex, EMV2) fogalmait követve.

Bár a teljes modellezési folyamat bemutatása túlmutat a tanulmány keretein, mindenképp érdemes kiemelni, hogy az EMV2 által definiált modellelemek több, egymásra épülő fázisban kerülnek meghatározásra, így maga a hibamodellezés is egy iteratív folyamat, amely vegyesen használ kvalitatív és kvantitatív módszereket. Az EMV2 az alap tervezői nyelvet kiegészíti egy, a biztonsági követelményeket és szempontokat, valamint a hibaterjedést leíró ontológiával (fogalomrendszerrel). Ez a kiegészítés lényegében azt támogatja, hogy a felhasználó az elkészített modellt „megjelölje” (szakszóval: annotálja) a hibamodellezés fogalmaival.

A többfázisú hibamodellezés illeszkedik egyébként ezeknek a rendszereknek a tanúsítási folyamataiba is, amelyek elvárnak egy előzetes rendszerelemzést, és egy, az elkészült rendszeren értelmezett, részletesebb elemzést is. Egy általános okos városi szolgáltatásnál természetesen jóval enyhébbek lehetnek a követelmények, mint egy repülésben használt rendszernél, de épp az elvek általánosíthatósága miatt ismertetjük az AADL-megközelítését. Fontos kiemelni, hogy az AADL által leírt modellezési kiegészítések, illetve vizsgálatok függetlenek a repülés fogalmaitól, és a legtöbb elem elérhető ingyenes, esetenként nyílt forráskódú eszközökben is. Példaként modellezésre alkalmas az OSATE (Carnegie Mellon 2006), hibafavizsgálatra a széles körben alkalmazott OpenFTA (OpenFTA), formális vizsgálatokra pedig a PRISM (Kwiatkowska–Norman–Parker 2011), a megközelítés használata így nem irreális egy kritikus okos városi példa esetén sem (lásd például a később bevezetendő intelligens kereszteződés példáját).

Dokumentum	Fázis	EMV2-modellelem (artifact)
Funkcionális veszélyek felmérése (Functional Hazard Assessment, FHA)	Előzetes rendszerszintű biztonságossági vizsgálat (Preliminary System Safety Assessment, PSSA)	Hibaforrások, hibaesemények
Hibafaelemzés (Fault Tree Analysis, FTA)	PSSA + Rendszerszintű biztonságossági vizsgálat (System Safety Assessment, SSA)	Összetett hibák definiálása, hibaesemények
Markovi analízis (MA), a rendszerben bekövetkező időbeli események vizsgálata	Rendszerszintű biztonságossági vizsgálat (System Safety Assessment, SSA)	Hibaforrások, komponensszintű hibamódok, hibavalószínűségek, hibaesemények
Hibamód és -hatásvizsgálat (FMEA)	Rendszerszintű biztonságossági vizsgálat (System Safety Assessment, SSA)	Hibaforrások, hibautak, hibákat elnyelő elemek (error sink)

4. táblázat. Rendszer ellenőrzési lépések és dokumentumok
Forrás: a szerzők saját szerkesztése

4.4.3. Üzleti folyamatok webszolgáltatás alapon: BPEL

A BPEL (Business Process Execution Language) a kifejezetten végrehajtható, XML-webszolgáltatás alapon működő folyamatok modellezését célozza meg.

A webszolgáltatások jellemzője, hogy elosztott, heterogén rendszerek funkcionalitása összekapcsolását támogatják. Nagyvállalati rendszerek összekapcsolásában az XML-webszolgáltatásokat használják arra, hogy a rendszerek közti kommunikációt megvalósítsák (például állami infrastruktúra szolgáltatásai, biztosítók/bankok szolgáltatásai, energiakereskedelem stb.). Mivel komplex üzleti folyamatokat tipikusan nem egy kérdés-válasz pár segítségével valósítanak meg, hanem valamilyen előírt kommunikációs minta mentén, szükséges ezen kommunikációs minta (feltételek kiértékelése mentén végrehajtott kérdés/válasz sorozatok) leírása. Ennek egyik eszköze a BPEL modellezési nyelv, amely egyben közvetlenül végrehajtható folyamatokat eredményez, ahol a folyamatvégrehajtás a különböző szereplők által kiajánlott (és így tipikusan különböző infrastruktúrákban működő) szolgáltatások meghívását jelenti.

A BPEL-nyelv elemkészlete miatt elsődlegesen fekete doboz jelleggel tud az egyes részlépések hibáira tekinteni, azonban a nyelv rendelkezik egyfelől hibakezelési, másrészt „kompenzációs” mechanizmusokkal, amelyek szisztematikus használatával az egyes hibák hatása csökkenthető, illetve a szokásos hibatűrő minták alkalmazhatók (Kovács–Gönczy 2008).

4.4.4. BPMN

Az üzleti folyamatot leíró nyelvek közül a leggyakrabban használt a Business Process Modelling Notation (BPMN). A BPMN rendelkezik a szokásosnak mondható esemény- és hibakezelési metódusokkal, ismeri a kivételkezelés fogalmát. A BPMN-nyelven leírt folyamatok, kiegészítve az egyes lépések megvalósítására utaló elemekkel (amelyet a szabvány támogat) szintén szolgálhatnak a hibaterjedési vizsgálatok bemenetéül (Csertán et al. 2002; Urbanics et al. 2014).

Érdemes kiemelni, hogy bármely modellezési nyelv, illetve módszertan esetén tipikusan a hibaterjedés és a hibafa jellegű elemzések kiegészítik egymást, iteratívan kombinálhatók, egyrészt követve a rendszer modellezése során a rendszer hierarchikus felépítését, másfelől a két módszer egymást kiegészítve bemenetként szolgálhat a következő elemzési fázisnak (lásd például Peeters–Basten–Tinga 2018).

5. HIBATERJEDÉSI VIZSGÁLAT „INTELLIGENS KERESZTEZŐDÉS” SZOLGÁLTATÁSAIBAN (ESETTANULMÁNY)

Az okos városi szolgáltatások egyik legfontosabb, rohamosan növekvő területe az okos közlekedés, illetve a közlekedésben részt vevő „okos járművek” támogatása. Ezen a területen kihívást jelent egyfelől a járművek intelligenciájának megvalósítása („önvezető járművek”), ami kívül esik jelen tanulmány témakörén. Kihívás ugyanakkor a közlekedés résztvevőinek kommunikációja is, valamint a közlekedés résztevőinek és az okos város infrastrukturális elemeinek kölcsönhatása.

Az alábbiakban a US Department of Transportation ajánlása (Arinc 2012) alapján példát mutatunk arra, milyen magas szintű követelmények alapján lehet egy megbízhatósági elemzést elvégezni a közlekedéssel kapcsolatos biztonságkritikus alkalmazások tervezésekor. Az önvezető gépjárművek elterjedésével és az ezek által okozott balesetek/incidensek elemzésekor kiemelt figyelmet kap a járművek megbízhatósága, amely azonban szinte sosem csak az adott járműben futó szoftver, illetve az azt kiszolgáló hardver tulajdonságaiból következik, hanem egyben jelentősen függ a környezeti tényezőktől, illetve a humán faktortól. Itt most az alkalmazással szemben támasztott követelményekkel foglalkozunk elsődlegesen.

A tipikus alkalmazás főbb komponensei:

- Fedélzeti számítógép (On Board Unit, On Board Equipment): a járműben található, korlátozott adatfeldolgozó és számítási képességekkel rendelkező számítógép. Feltételezzük, hogy a számítógépen futó programok és azok beállításai frissíthetők, de ehhez adott esetben szakértői beavatkozás kell.
- Járművek érzékelői/szenzorai: például RADAR/LIDAR/Ultrahangos szenzorok, kamerák, melyek a jármű távolságmérését segítik. Bár nem szokták kiemelni külön (hiszen egyébként is részét képezi a járműnek), de ide érthető a sebesség, hőmérséklet, eső szenzor is. A most elérhető specifikációk ezzel tipikusan nem foglalkoznak, de adott esetben például a módosult fékút vagy kanyarodási képesség miatt a jármű lokális jelzéseit is érdemes lehet beépíteni az alkalmazásokba akár oly módon is, hogy (megfelelő validálás mellett) egy adott jármű észlelése más járműveknek adott jelzésnél is felhasználható legyen. Ez természetesen a kereszteződés vezérlését végző komponensre nézve ellenőrzési és kommunikációs pluszfeladatokat jelent, illetve természetesen újabb hibamódokat hozhat be.
- Járművek kommunikációs egységei: a járművekbe épített, elsődlegesen rádiós, mobilhálózati, illetve WiFi-kommunikációt támogató egységek.
- Közlekedési lámpa (Traffic Light): Az intelligens közlekedési lámpa ebben az esetben a kereszteződés „vezérlését” is végzi, amennyiben nemcsak a forgalmi jelzéseket vezérli, hanem adott kommunikációs protokollok mellett mind a központi informatikai rendszerrel, mind a járművekkel információt cserél.

- Központi informatikai rendszer: Kiberfizikai rendszerekben szokásos módon a központi intelligencia (amely lehet akár felhőalapú alkalmazások összessége) gondoskodik a lokálisan rendelkezésre nem álló információk frissítéséről. Ilyen lehet például a térképpel kapcsolatos információ, járművek besorolása, veszélyhelyzetek, preferenciák stb.
- További érzékelők: Az infrastruktúrához kapcsolódó további, áthaladást, távolságot stb. mérő érzékelők, kamerák.

Az ajánlás által tárgyalt alkalmazások alapvető funkcionalitása a következő:

Tilos jelzésen áthaladás megelőzése (Red Light Running, RLR). A tilos jelzésbe érkezést megelőző alkalmazás célja a vezető figyelmeztetése, hogy adott körülmények mellett (adott távolságban, adott sebességnél, ismerve a közlekedési jelzések ütemezését) a gépjármű nagy eséllyel „piros” jelzésben fog egy adott kereszteződéshez érni. Az alkalmazások tipikusan mind a fix, mind a változó irányítási ütemezésű lámpák kezelésére (például gyalogos vagy ritkán közlekedő jármű által kezdeményezett váltás) fel vannak készítve. Az alkalmazásnak tehát az a fő kritériuma, hogy legkésőbb adott távolságnál kellően biztos döntést tudjon hozni arról, hogy veszélyhelyzetet okoz-e, ha a sofőr fenntartja a sebességet.

Balra kanyarodás támogatása (Left Turn Assist, LTA). Célja a balra kanyarodás támogatása (feltételezve a hagyományos „kontinentális” balkormányos, jobb oldali közlekedési rendet) oly módon, hogy figyelmeztetést ad a sofőr számára, ha (becsült további sebesség mellett) balra akar kanyarodni adott kereszteződésben. Az alkalmazás megvalósítható mind a közlekedési infrastruktúrát használó, mind csak a járművek közti (Vehicle2Vehicle) kommunikációra építő módon, előbbi közvetlenül épít az „okos város” szolgáltatásaira.

Az alkalmazás által adott kimenet felismeri, ha a) a jármű biztonságosan be tud kanyarodni, mert a szembejövő jármű(vek) a kereszteződés előtt lesznek a kanyarodás pillanatában, b)a jármű biztonságosan be tud kanyarodni, mert a szembejövő forgalom áthalad a kereszteződésen a kanyarodás pillanatáig, c)a kanyarodás nem biztonságos, mert a kereszteződésben várhatóan jármű halad át a kanyarodás pillanatában. Előbbi két eset ugyanahhoz a kvalitatív kimeneti osztályhoz („Nincs veszély”) tartozik, míg utóbbi a „Veszély/Ne kanyarodj!” kimenetnek felel meg.

Jobbra kanyarodás támogatása (Right Turn Assist, RTA): Nagyban megfelel a balra kanyarodás támogatásának, de nem a kereszteződésben szembejövő, hanem a balról érkező járműveket figyeli (megjegyzés: ezzel feltételezve a szembejövő szabályos közlekedését, ami rögtön felveti a hibamodell teljességének kérdését). A kimenetei is hasonlók, viszont annyiban kritikusabb lehet a hibamodell, hogy a balról érkező járművek a vezető számára adott esetben nem láthatók, így az alkalmazás téves „Nincs veszély” jelzése még inkább kritikus, ezt érdemes a hibaelemzésnél figyelembe venni.

Mindkét kanyarodást támogató alkalmazás képes lehet arra is (amennyiben az okos városi infrastruktúra ezt támogatja), hogy a kanyarodás során érintett gyalogosátkelők forgalmát is figyelembe vegye.

Vasúti átjáró tilos jelzésére figyelmeztető alkalmazás (Rail Crossing Red Light Violation, RCRLV): Hasonló a tilos jelzésen való áthaladásra figyelmeztető alkalmazáshoz, ez esetben értelemszerűen lámpával jelzett vasúti átkelőre értelmezve.

Megkülönböztető jelzést használó jármű megelőző beavatkozása (Emergency Vehicle Preemption, PREEMPT). A megkülönböztető járművek kereszteződésen történő áthaladása rendszeresen okoz baleseteket, illetve veszélyhelyzeteket, amelyek elkerülésének egyik lehetséges módja, ha a jármű előzetesen értesíti az intelligens kereszteződést,

hogy át fog haladni, így az vagy egy zöld jelzés időtartamát növeli meg (ha a jármű zöld helyett már számára piros jelzésben érne a kereszteződésbe), vagy egy piros jelzés időtartamát csökkenti (és egyben a keresztbe haladó forgalom számára lerövidíti a zöldet).

Ennek az alkalmazásnak a hibamodelljében kiemelt szerepet kap az időzítés, ugyanis a lámpa normál rendjébe történő beavatkozás miatt hatása implicit módon megjelenik más alkalmazásokban, így adott esetben kaszkádhibához vezethet. Ha például az alkalmazás rosszul becsüli meg, hogy a megkülönböztető jelzést használó jármű a kereszteződésen biztonságosan át tud haladni, akkor az egyben a keresztirányú forgalom intelligens járműveiben futó, a kereszteződésen való áthaladás biztonságosságát jelző alkalmazás „Biztonságos" jelzése ebben az esetben „hamis negatív" lesz, ami fokozza a baleset bekövetkezésének esélyét.

Tömegközlekedési járművek prioritását támogató alkalmazás (Transit Signal Priority, TSP): Az alkalmazás célja némileg hasonló, mint a megkülönböztető jelzést használó járműveket támogató alkalmazás esetében: a közlekedés egy szereplőjét (jelen esetben az utasszállítást végző buszokat, kisbuszokat) a globálisan kedvezőbb/hatékonyabb közlekedési szolgálatás érdekében indokolt esetben előnyhöz juttatni azáltal, hogy adott forgalmi szituációban a zöld jelzést meghosszabbítja a kereszteződés vezérlője.

Az alkalmazás érdekessége, hogy elvárható a busz által adott prioritáskérő üzenetben a prioritás kérésének indoklása (például 5 percnél nagyobb késés a tervezett ütemezéshez képest). Noha nem feltétlenül része az alapalkalmazásnak, egy integrált okos városi környezetben akár lekérdezhető mindehhez a menetrendi adat, vagy ellenőrizhető a kérés a közlekedési szolgáltató rendszerébe küldött üzenettel. Ekkor azonban a hibaterjedési vizsgálat fontos kérdése lehet, hogy az adott esetben nem kritikus, valós idejű használatra felkészített közlekedési/forgalmi lekérdezést támogató alkalmazások nem jelentenek-e egy új, potenciális, külső hibaok-forrást.

Tehergépjárművek prioritását támogató alkalmazás (Freigth Signal Priority, FSP): Hasonlóan a tömegközlekedési járművek alkalmazásaihoz, ennek is az a célja, hogy egy arra a rendszer által jogosultnak talált jármű áthaladása érdekében a szabad jelzés időtartamát módosítsa. Így biztonságosabb és takarékosabb forgalmi helyzetek állhatnak elő, hiszen kevesebbet kényszerülnek fékezésre a nagy tömegű járművek.

Ezen alkalmazások közösen az ún. SPAT (Signal Phase and Timing) szabványt használják. Ehhez a szabványhoz tartozik egy olyan táblázat, amely egyben a fenti alkalmazások kommunikációs hibamodelljéhez is bemenet lehet.

A táblázat a kereszteződésekben támogató biztonságkritikus alkalmazásokra adja meg a kereszteződéstől mért minimumtávolságot, amiben az alkalmazásnak megfelelően kell működnie. Mindez értelemszerűen függ a gépjármű sebességétől, hiszen a vezető reakcióideje, illetve magának az alkalmazásnak a válaszideje mint összesített idő alatt a gépjármű jelentősen közeledhet a kereszteződéshez. Vegyük figyelembe, hogy a sebességek a tipikus amerikai sebességhatárokhoz vannak igazítva (30 mph kb. 48 km/h). Itt külön kiemelendő, hogy adott esetben nem minden alkalmazás működik minden sebességtartományban, tehát egy tervezési időben eldöntendő kérdés, hogy egy alkalmazást milyen tartományra kell felkészíteni (és adott esetben tanúsítani). A tartományok határai értelemszerűen összefüggghetnek az adott ország/régió sebességkorlátozásaival, tehát itt is érdemes lehet a Jakab–Kocsis–Gönczy 2018 monográfiában bemutatott kvalitatív absztrakciós módszer alkalmazása.

Ez a fajta modellezés egyben alkalmas arra is, hogy egy adott komponens (például jeladó) teljesítményét kiértékeljük, ha a komponenst használó járművet vagy rögzített infrastruktúraelemet a későbbiekben újabb alkalmazások támogatására szeretnénk felkészíteni.

Alkalmazás\elvárt kommunikációs távolság	Sebességtartomány, mérföld/óra (km/h)			
	30 (48)	45 (72)	60 (96)	75 (120)
Tilos jelzésen történő áthaladás megelőzése (Red Light Running, RLR)	36 (méter)	72	118	176
Balra kanyarodás támogatása (Left Turn Assist, LTA)	36	72	118	176
Jobbra kanyarodás támogatása (Right Turn Assist, RTA)	36	72	118	176
Vasúti átjáró tilos jelzésére figyelmeztető alkalmazás (Rail Crossing Red Light Violation, RCRLV)	36	72	118	176
Megkülönböztető jelzést használó jármű megelőző beavatkozása (Emergency Vehicle Preemption, PREEMPT)	197	331	--	--
Tömegközlekedési járművek prioritását támogató alkalmazás (Transit Signal Priority, TSP)	60	122	--	--
Tehergépjűrművek prioritását támogató alkalmazás (Freigth Signal Priority, FSP):	60	122	--	--

5. táblázat. Kommunikációs követelmények Smart City alkalmazásokhoz
Forrás: a szerzők saját szerkesztése ARINC 2012 alapján

A fenti táblázat egyben mutatja azt is, hogy ha egységes kvalitatív modellt készítünk a távolság tartományokról, akkor a későbbiekben egy hibamodellezés során jelentősen egyszerűsödhet (például ha egyszer beláttuk, hogy a lámpa kommunikációs egységének hibája elhanyagolható a „kis” távolságokra, ahol minden 120 méternél kisebb távolság kicsi, akkor ezzel számos alkalmazás esetében megfelelhetünk a követelményeknek).

5.1. A tilos jelzésen áthaladás (Red Light Running) alkalmazás hibamodellje és elemzése

Az alkalmazás alapvetően négy kimenetet adhat (feltételezve a megszokott piros-sárga-zöld közlekedési lámpát, azzal a kikötéssel, hogy a leginkább elterjedt zöld-sárga-piros sorrendben vált a lámpa a „szabad áthaladás” felől az „áthaladni tilos” irányba):

- Nincs figyelmeztetésre szükség (No Warning): A lámpa zöld jelzést ad, és a sárga jelzés bekövetkezése a kereszteződésen történő áthaladás utánra várható.
- Figyelmeztetés (Caution): A lámpa zöld jelzést ad, de várhatóan sárgára vált, miközben a jármű áthalad a kereszteződésen.
- Kiemelt figyelmeztetés (Extreme Caution): A lámpa sárga jelzést ad, és várhatóan pirosra vált, miközben a jármű áthalad a kereszteződésen.
- Stop!: A lámpa sárga jelzést ad, és várhatóan pirosba vált, mielőtt a jármű áthaladna a kereszteződésen.

Megjegyzés: az alkalmazás a nevéből is adódóan és a biztonsági követelményekből értelemszerűen következő módon nem foglalkozik azzal, hogy a közlekedést azáltal optimalizálja, hogy a lámpa piros-zöld átmenetét próbálja becsülni, és az esetleges felesleges fékezéseket/lassításokat elkerülni azzal, hogy a kereszteződés szabaddá válását becsli meg.

Amennyiben a fenti alkalmazáshoz FMEA-elemzést szeretnénk elvégezni, rögtön megállapítható, hogy az egyes hibák kritikussága eltérő lesz attól függően, milyen irányba téved az alkalmazás.

Kiegészítve az ajánlást, megalkottuk az alkalmazás hibamodelljét és ebből adódóan az egyes algoritmikustévedések egyik lehetséges besorolását:

Valós \ Becsült	Nincs figyelmeztetés (No Warning)	Figyelmeztetés (Caution)	Kiemelt figyelmeztetés (Extreme Caution)	Stop!
Nincs figyelmeztetés (No Warning)	Jó eredmény (0)	1	2	2
Figyelmeztetés (Caution)	1	Jó eredmény (0)	1	2
Kiemelt figyelmeztetés (Extreme Caution)	2	2	Jó eredmény (0)	1
Stop!	3	3	3	Jó eredmény (0)

6. táblázat. Besorolás tévedéseinek súlyossága
Forrás: a szerzők saját szerkesztése

A fenti mátrix megfelel egyben annak a formának, amellyel egy-egy adott komponensben futó döntési algoritmusokat értékelni szoktak. Az ún. konfúziós mátrix lényege, hogy egy adott, osztályokba sorolást végző kiértékelés eredményénél a valós és a becsült értékek eltérését mutatja meg (például hány olyan „Not Warning” szituáció volt, amelyet a szoftver helytelenül „Stop!” szituációnak sorolt be, vagyis veszélytelen helyzetben vészfékezésre késztette volna a sofőrt).

A fenti táblázat az ajánlás alapján egy lehetséges súlyozása az egyes hibajelenségeknek, vagyis a hibás kategorizálásnak, egy feltételezett 0–3 skálán, ahol a 3 a katasztrofális, 2 a veszélyes, 1 az enyhe hiba. A súlyozás maga természetesen változhat, viszont adott súlyozás egyben támpontot is szolgáltat a hibaterjedés egyes kimeneteinek értékeléséhez.

A táblázat kitöltésekor azt az alapelvet vehetjük figyelembe, hogy a kritikus szituáció fel nem ismerése mindenképp kerülendő, tehát a lehető legsúlyosabb tévedésnek tekintjük minden esetben, míg azokban az esetekben, ahol a rendszer felesleges leállást jelez, jelentősnek tekintjük a hibát, mivel felesleges zavart és veszélyhelyzetet okozhat.

Utóbbi feltételezés annak felel meg, hogy a többi közlekedő helyesen ismeri fel a szituációt, tehát nincs „közös módusú hiba”. Mivel a rendszer (megvalósítástól függően) adott esetben magával a közlekedési lámpával kommunikál, ez nem feltétlenül igaz, azonban az analízis szempontjából egy jó felső (pesszimista) közelítés.

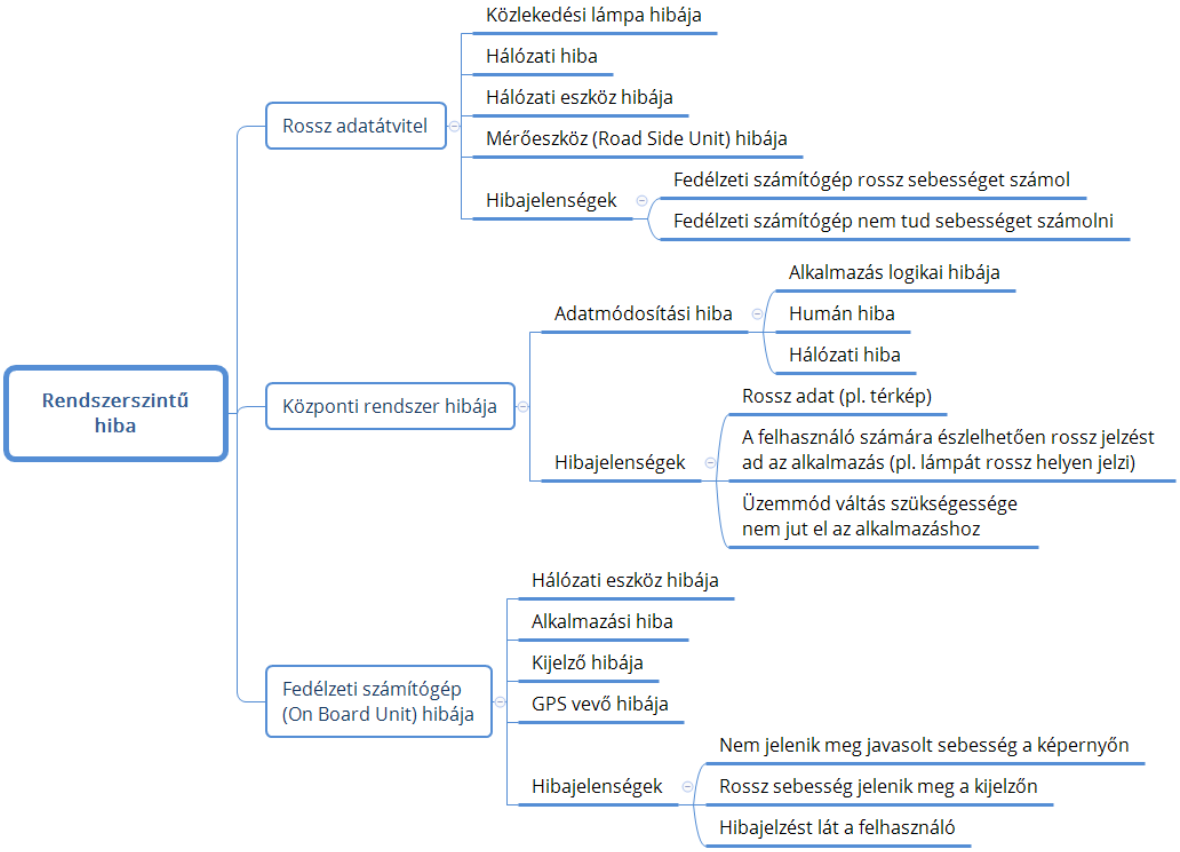
Amennyiben a hibaterjedést az FMECA (Failure Mode, Effect and Criticality Analysis) elvei szerint végezzük el, ez az értékelés egyben a különböző kimenetek súlyosságát is megadja. Megfordítva, amennyiben azt keressük, hogy milyen bemenetek, illetve rendszerállapotok kombinációja vezethet el ahhoz, hogy a rendszer súlyos hibát vétsen, szintén felhasználhatók azok a kimenetek, melyek a „Stop!” állapotot detektálják rosszul.

A fenti gondolatmenet egyben arra is utal, hogy a terjedőben lévő mesterséges intelligencia alapú módszerek kimenetének értékelésekor és megbízhatósági elemzésekor a szakértői súlyozás elengedhetetlen, mivel az egyes tévedések súlya nem azonos.

Mennyiben segíti egy ehhez hasonló, magas szintű követelményeket tartalmazó táblázat a szolgálatás/rendszer megfelelőségének vizsgálatát?

A vizsgálat során a sebesség és a hiba súlyossága mellett egyéb kvalitatív változók is értelmezhetők, például a távolság. Ahogyan bevezettük, az ajánlást megkülönbözteti a névleges (176 méteren belüli) és a távoli (331 méteren belüli) üzenetfogadást mint műveletet az üzenetvesztés valószínűsége szempontjából.

A fenti módszer gyakorlati alkalmazására ad példát Nagel 2015. A munkában elkészült egy fedélzeti driving assist szolgáltatást nyújtó rendszer, amely a fentiek közül a tilos jelzésen történő áthaladásra figyelmeztet. A mű példája alapján, annak modelljét kiegészítve, bemutatjuk a RLR-alkalmazás egy lehetséges hibataxonómiáját.



4. ábra. Egy intelligens kereszteződésre épülő alkalmazás lehetséges hibamodellje
Forrás: a szerzők saját szerkesztése

A hibamodell tovább finomítható, elsősorban az algoritmikával/szoftveres vezérléssel összefüggő hibáknál. Például a már említett konfúziós mátrix segítségével kiértékelhetjük, hogy milyen kommunikációs/mérési problémák lesznek kritikusak, illetve pl. egy rádiós protokoll adott hibamódjai közül melyik vezethet el az alkalmazás kritikus hibájáig.

A fenti hibamódok is bemutatják, hogy az okos városi alkalmazásfejlesztés során az egymással együttműködő elemek miatt a megbízhatósági elemzés túlmutat az egyes szakterületeken (például az autóipari alkalmazások fejlesztésén) belül elvégezhető vizsgálatok hatókörén. Hozzátehetjük azt is, hogy jelen esetben csak egy-egy alkalmazás hibájáról beszéltünk, de felmerül a kérdés, hogy az egyes alkalmazásokat közösen kezelő elemeknél logikai milyen egymásra hatása lehet az alkalmazásoknak. Ez alatt

nem csak a közös módusú hibákat kell érteni (például tegyük fel, egy rosszul megírt alkalmazás miatt elfogy a kereszteződés vezérlő egységének memóriája), hanem például a prioritások kezelését is: mi történik, ha egyszerre kérne áthaladást egy merőleges útkereszteződésbe érkező mentő rohamkocsi és egy tehergépjármű? Mi történik két megkülönböztető jelzést használó jármű esetében? Mi történik, ha egy járműnek visszaigazolta a rendszer a lehetséges áthaladást, de közben nagyobb prioritású jármű érkezik, stb.?

5.2. A kritikus besorolás hatása a rendszer fejlesztésére

Itt érdemes utalni az autóiparban elterjedt, ASIL biztonság integritási szintekre, melyek egyben megszabják egy adott alkalmazás kritikusságát. Ezen biztonságintegritás-szintek megszabják egy alkalmazás fejlesztésével, ellenőrzésével (ideértve mind a kimerítő, formális módszerekkel történő ellenőrzést, mind a szakértői vizsgálatokat, mind a tesztelést) kapcsolatos folyamatokat. A hibamodellezésnél mind az egyes komponensek hibamódjainál, hibatűrési tulajdonságainál, mind az elemzés kiértékelésénél érdemes figyelembe venni ezt a fajta besorolást, mert mind az alkalmazással/komponenssel szemben támasztott fejlesztési, ellenőrzési és üzemeltetési elvárások, mind a teljes rendszerrel szemben támasztott követelmények erősen függenek a besorolástól.

Ezért az ASIL-besorolás egyben jó példája a kvalitatív követelmények kiértékelésének, ahogyan azt az AUTOSAR szabvány és Soden 2011, valamint Gulland 2004 alapján az alábbi táblázat is mutatja. A QM (Quality Managed) alkalmazások esetében jóval enyhébbek a követelmények, és tanúsítás sem szükséges.

		1. Kontrollálhatóság	2. Kontrollálhatóság	3. Kontrollálhatóság
1. Súlyosság	1. Kitettség	Minőségbiztosított (Quality managed, QM)	Minőségbiztosított (Quality managed, QM)	Minőségbiztosított (Quality managed, QM)
	2. Kitettség	QM	QM	QM
	3. Kitettség	QM	QM	ASIL A
	4. Kitettség	QM	ASIL A biztonság integritási szint	ASIL B
2. Súlyosság	1. Kitettség	QM	QM	QM
	2. Kitettség	QM	QM	ASIL A
	3. Kitettség	QM	ASIL A	ASIL B
	4. Kitettség	ASIL A	ASIL B	ASIL C
3. Súlyosság	1. Kitettség	QM	QM	ASIL A
	2. Kitettség	QM	ASIL A	ASIL B
	3. Kitettség	ASIL A	ASIL B	ASIL C
	4. Kitettség	ASIL B	ASIL C	ASIL D

7. táblázat. ASIL-kockázati szintek
Forrás: a szerzők saját szerkesztése az AUTOSAR szabvány, SODEN 2011 és GULLAND 2004 alapján

A fenti szempontok az alábbiak:

Severity/Súlyosság: az adott funkció hibája milyen súlyosságú hibát okoz (ARINC 2012 kiterjesztésével).

Súlyosság/ Severity			
S0	Nem várható sérülés	Nem várható sérülés	Például alacsony alatti sebességgel nekitolat a jármű egy akadálynak
S1	Könnyeb sérülések	>10% valószínűsége az 1-6 AIS sérüléseknek (feltéve, hogy S2 és S3 nem áll fenn)	Közepes sebességű tolatásnál bekövetkezett baleset.
S2	Súlyos és életveszélyes sérülések (lehetséges a túlélés)	>10% valószínűséggel 3-6 AIS hatás (de nem S3)	Például városi közlekedésben gyalogos/ kerékpáros gázolás
S3	Halálos és életveszélyes sérülések (a túlélés bizonytalan)	>10% valószínűséggel AIS 5-6 szintű hatás	Például országúti gyalogos/ kerékpáros gázolás

8. táblázat. Súlyosság és következmények
Forrás: a szerzők saját szerkesztése az AUTOSAR szabvány, SODEN 2011 és GULLAND 2004 alapján

(Megj.: Az AIS-skála, az Abbreviated Injury Scale a becsült személyi sérüléseket próbálja meg – bár félrevezető módon számokkal kódolva, de – kvalitatív módon közelíteni, 1–6 közt. A nominális skálával történő közelítés ugyan nyilvánvalóan elvi hibás, hiszen nem értelmezhetők az aritmetikai műveletek, ugyanakkor jól kifejezi a kvalitatív megközelítés és az intuitív szakértői gondolkodás hasonlóságát.)

Exposure/Kitettség: az adott hibaoknak milyen gyakran van kitéve a rendszer, lásd az alábbi tartományokat:

Exposure/ Kitettség			
E0	Elhanyagolható	A működési idő kevesebb, mint 0.01%-a.	Például Természeti katasztrófák.
E1	Nagyon alacsony valószínűségű	A működési idő kevesebb, mint 0.1%-a. (Olyan helyzetek, melyek egy átlagos sofőrnél évente max. egyszer fordulnak elő.)	Például leáll a motor egy kereszteződésben.
E2	Alacsony valószínűség	A működési idő kevesebb, mint 1%-ában előforduló helyzetek (a legtöbb sofőrnél évente legfeljebb néhány alkalom).	Például korlát nélküli hegyi úton vezetés, ismeretlen útvonalon.
E3	Közepes valószínűség	A működési idő kevesebb, mint <10%-ában fordulnak elő. Egy tipikus sofőr havonta néhány alkalommal kerül ilyen helyzetbe.	Nedves úton vezetés, előzés, tankolás stb.
E4	Magas valószínűség	10%-nál gyakrabban fordul elő (minden tipikus vezetésnél).	Fékezés, parkolás, irányváltás stb.

9. táblázat A kitettség osztályozása és következményei
Forrás: a szerzők saját szerkesztése az AUTOSAR szabvány, SODEN 2011 és GULLAND 2004 alapján

Vezérelhetőség/Controllability: az adott eseményt kiváltó veszélyhelyzetet milyen esélylyel tudja elkerülni a jármű vezetője, mennyire tudja az adott helyzetet kezelni. Tehát ez a tulajdonság alapvetően arra vonatkozik, hogy a rendszeren kívülről, a felhasználótól érkező input mekkora eséllyel segíthet a veszélyhelyzet (hazard) eredményeként bekövetkező esemény (event) elkerülésében.

Vezérelhetőség/ Controllability			
C0	Általánosan kontrollálható	Általában kézbentartható esemény	Például a rádió hangereje hirtelen megváltozik
C1	Egyszerűen kontrollálható	A közlekedésben résztvevők 99% el tudja kerülni a helyzetet	Például a jármű rögzített kormánnyal indulna el.
C2	Normális esetben kontrollálható	A vezetők vagy résztvevők 90%-a kézben tudja tartan a helyzetet.	Például vészfékezésnél az ABS hibája esetén a sofőrt el tudja kerülni a sáv hirtelen elhagyását.
C3	Bonyolult vagy kézbentarthatatlan	A közlekedésben résztvevők kevesebb, mint 90%-a tudja elkerülni a helyzetet	Ha a féknyomás „elmegy”, a sofőr nem tudja megállítani a járművet.

10. táblázat. A vezérelhetőség osztályozása és következményei
Forrás: a szerzők saját szerkesztése az AUTOSAR szabvány, SODEN 2011 és GULLAND 2004 alapján

A következményekre az alábbi kvalitatív tartományok definiálhatók:

- A: kisebb sérülést okozhat.
- B: A halálos áldozatok várható száma 0,01–0,1 közti.
- C: A halálos áldozatok várható száma 0.1 és 1 közt van.
- D: Több mint 1 halálos áldozat várható.

5.3. Az alkalmazás/szolgáltatás és az infrastruktúra kritikusságának egymásra hatása

A fentiekben példát mutattunk arra, hogy egy összetett okos városi alkalmazás védelme – értve ez alatt akár csak a nem szándékos hibák elleni védelmet, a biztonsági (security) aspektusok nélkül – a zárt rendszereken túlmutató elemzést igényel. Mindezeken túl, mivel a szereplők közti kommunikáció és információcsere mértéke és jelentősége rohamosan bővül (példaként az intelligens, a környezettel, az infrastruktúrával és adott esetben a többi járművel kommunikáló járműveket tekintve), a rendszerek közti egymásra utaltság modellezése és a komponensek közti dinamikus hibaterjedés vizsgálata egyre nagyobb jelentőséget kap. Abban az esetben például, ha a különböző „driving assist" szolgálatások egyre növekvő mértékben befolyásolják a sofőrök döntéseit, egy esetleges váratlan helyzet vagy téves működés egyre nagyobb hibát tud okozni.

Ez megfelel annak, hogy a fenti besorolásban várhatóan most C0, C1 (kontrollálható) helyzetek akár felsőbb kategóriákba is kerülhetnek, ha a sofőrök a szolgáltatások helyes működéséhez hozzászokva azok lehetséges meghibásodásaira rosszabbul reagálnak, mint ezen alkalmazások használata nélkül. Példa lehet erre akár a kereszteződésen történő áthaladás, akár a sávtartást, akár a parkolást segítő rendszerek hibája, Ezek az alkalmazások ugyanakkor nemcsak az adott jármű mint rendszer tervezője által kézben tartható szempontoktól, hanem az intelligens, adatot szolgáltató infrastruktúra tervezője és üzemeltetője által befolyásolható tényezőktől függenek (például a lámpa által szolgáltatott adat pontossága és kiszámíthatósága).

IRODALOMJEGYZÉK

A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr (2004), Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11–33, IEEE

B. Bittner et al. (2016), The xSAP Safety Analysis Platform, in Tools and Algorithms for the Construction and Analysis of Systems, LNCS 9636, pp. 533–539, Springer

V. Bonfiglio, L. Montecchi, F. Rossi, P. Lollini, A. Pataricza and A. Bondavalli (2015), Executable Models to Support Automated Software FMEA, in 2015 IEEE 16th International Symposium on High Assurance Systems Engineering (HASE), pp. 189–196, IEEE

A. Bondavalli and L. Simoncini (1990), Failure classification with respect to detection, in Second IEEE Workshop on Future Trends of Distributed Computing Systems, pp. 47–53, IEEE

S. Bozóki, G. Koronka and A. Pataricza (2015), Risk Assessment Based Cloudification, in SERENE 2015: Software Engineering for Resilient Systems, pp. 71–81, Springer

M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll and M. Roveri (2011), Safety, Dependability and Performance Analysis of Extended AADL Models, Comput J, vol. 54, no. 5, pp. 754–775, IEEE

Carnegie Mellon Software Engineering Institute (2006), Open Source AADL Tool Environment - http://www.aadl.info, Technical Report

Gy. Csertán, György, A. Pataricza, P. Harang, O. Dobán, G. Biros, A. Dancsecz and F. Friedler (2002), BPM based robust E-Business application development, in European Dependable Computing Conference EDCC-4, pp. 32–43, Springer

P. David, V. Idasiak, F. Kratz (2010), Reliability study of complex physical systems using SysML, Reliability Engineering & System Safety, Volume 95, Issue 4, pp. 431-450, Elsevier

J. Delange and P. Feiler (2014), Architecture Fault Modeling with the AADL Error-Model Annex, in 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 361–368, IEEE

I. Dobson, J. Chen, J. S. Thorp, B. Carreras and D. Newman (2002), Examining criticality of blackouts in power system models with cascading events, in HICSS. Proceedings of the 35th Annual Hawaii International Conference on System Sciences, IEEE

M. Faida, N. Nguyen, H. Kadima and J.-Y. Choley (2013), Safety analysis integration in a SysML-based complex system design process, in 2013 IEEE International Systems Conference (SysCon), I, pp. 70-75, IEEE

Failure mode, effects and criticality analysis, https://en.wikipedia.org/wiki/Failure_mode,_effects,_and_criticality_analysis Utolsó hozzáférés: 2018. március 23.

P. H. Feiler and D. P. Gluch (2012), Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language, Addison-Wesley

B. Gallina and S. Punnekkat (2011), FI4FA: A Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures' Analysis, in 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 493–500, IEEE

X. Ge, R. F. Paige and J. A. McDermid (2009), Probabilistic Failure Propagation and Transformation Analysis, in Computer Safety, Reliability, and Security, pp. 215–228, Springer

J. R. Gil-Garcia, T. A. Pardo and T. Nam (2015), What makes a city smart? Identifying core components and proposing an integrative and comprehensive conceptualization, Information Polity, vol. 20, no. 1, pp. 61–87, IOS Press

D. R. Godschalk (2003), Urban Hazard Mitigation: Creating Resilient Cities, Natural Hazards Review, vol. 4, no. 3, pp. 136–143, American Society of Civil Engineers

L. Gönczy (2018), Kritikus infrastruktúrák és szolgáltatások modellezése, BME-NKE Okosváros kutatóműhely, kismonográfia. Megjelenés alatt.

L. Gönczy, S. Chiaradonna, F. Di Giandomenico, A. Pataricza, A. Bondavalli and T. Bartha (2006), Dependability Evaluation of Web Service-Based Processes, in Horváth A., Telek M. (eds), Formal Methods and Stochastic Models for Performance Evaluation. EPEW 2006. Lecture Notes in Computer Science, vol 4054. Springer

W.G. Gulland (2004), Methods of determining safety integrity level (SIL) requirements - pros and cons, in Practical Elements of Safety, pp. 105-122. Springer

R. Hanmer (2010), Software Rejuvenation, in Proceedings of the 17th Conference on Pattern Languages of Programs, pp. 21:1–21:13

R. Hanmer (2013), Patterns for Fault Tolerant Software. John Wiley & Sons
IEEE 1044-2009, IEEE Standard Classification for Software Anomalies (2009), Institute of Electrical and Electronics Engineers, IEEE Computer Society

IEEE 1633-2016, IEEE Standard Recommended practice on spwtware reliability (2016), Institute of Electrical and Electronics Engineers, IEEE Reliability Society

L. Jakab, I. Kocsis, L. Gönczy (2018), Kvalitatív módszerek és elemzési megközelítések felhasználása kritikus infrastruktúrák védelmében és teljesítménymenedzsmentjében I. rész, BME-NKE Okosváros kutatóműhely, kismonográfia. Megjelenés alatt.

G. Jiang, H. Chen, C. Ungureanu and K. Yoshihira (2007), Multiresolution Abnormal Trace Detection Using Varied-Length n-Grams and Automata, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 37, no. 1, pp. 86–97, IEEE

I. Kocsis (2018), Design for Dependability Through Error Propagation Space Exploration, in 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 172–178, IEEE

I. Kocsis (2018) IT rendszerek teljesítménymenedzsmentjének céljai, stratégiái és megközelítési módszerei, a kritikus szolgáltatások teljesítményének tervezése, BME-NKE Okosváros kutatóműhely, kismonográfia. Megjelenés alatt.

M. Kovács and L. Gönczy (2008), Simulation and formal analysis of workflow models, Electronic Notes in Theoretical Computer Science 211: 221–230, Springer

M. Kwiatkowska, G. Norman and D. Parker (2011). PRISM 4.0: Verification of probabilistic real-time systems, in Proc. 23rd International Conference on Computer Aided Verification (CAV'11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806, pp. 585–591, Springer

R. E. Lyons and W. Vanderkulk (1962), The Use of Triple-Modular Redundancy to Improve Computer Reliability, IBM Journal of Research and Development, vol. 6, no. 2, pp. 200–209, IBM

I. Majzik, A. Pataricza and A. Bondavalli (2003), Stochastic Dependability Analysis of System Architecture Based on UML Models, in de Lemos R., Gacek C., Romanovsky A. (eds) Architecting Dependable Systems. Lecture Notes in Computer Science, vol 2677, Springer

C. E. Nagel (2015), Design and Failure Mode and Effects Analysis (FMEA) of a Vehicular Speed Advisory System, MSc thesis, Hamburg University of Technology

P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano and F. Scorrano (2014), Current trends in Smart City initiatives: Some stylised facts, Cities, vol. 38, pp. 25–36, Elsevier

A. Pataricza (2008), Model-based dependability analysis, DSc thesis, Hungarian Academy of Sciences

A. Pataricza (2008), Systematic generation of dependability cases from functional models, in Proc. of the FORMS/FORMAT Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, L'Harmattan

J. F. W. Peeters, R. J. I. Basten and T. Tinga (2018), Improving failure analysis efficiency by combining FTA and FMEA in a recursive manner, Reliability Engineering & System Safety, Volume 172, pp. 36–44, Elsevier

R. Pietrantuono and S. Russo (2013), Introduction to Safety Critical Systems, in Innovative Technologies for Dependable OTS-Based Critical Systems, pp. 17–27, Springer

S. pilot (2002), What is a fault tree analysis?, http://asq.org/quality-progress/2002/03/problem-solving/what-is-a-fault-tree-analysis.html

G. Pintér, H. Madeira, M. Vieira, I. Majzik and A. Pataricza (2005), A Data Mining Approach to Identify Key Factors in Dependability Experiments, in Dependable Computing - EDCC 5, 2005, pp. 263–280, Springer

G. Pinter, H. Madeira, M. Vieira, I. Majzik, and A. Pataricza (2008), Integration of OLAP and data mining for analysis of results from dependability evaluation experiments, International Journal of Knowledge Management Studies, vol. 2, no. 4, pp. 480–498, Inderscience Publishers

R. Rajkumar, I. Lee, L. Sha and J. Stankovic (2010), Cyber-physical Systems: The Next Computing Revolution, in Proceedings of the 47th Design Automation Conference, pp. 731–736, IEEE

Reliability, Availability and Maintainability: System
Engineering Book of Knowledge: http://sebokwiki.org/wiki/Reliability,_Availability,_and_Maintainability Utolsó hozzáférés: 2017. augusztus 25.

M. Soden (2011), ISO 26262 for Safety-Related Automotive E/E Development - Introduction and Concept Phase, ISO

P. Szeredi and T. Benkő (2002), Nagyhatékonyságú logikai programozás, Budapest, BME Számítástudományi és Információelméleti Tanszék

J.C. Tan, P. A. Crossley, P. G. McLaren, P. F. Gale, I. Hall and J. Farrell (2002), Application of a wide area backup protection expert system to prevent cascading outages, IEEE Transactions on Power Delivery, 17, no. 2, 375–380, IEEE

G. Urbanics, L. Gönczy, B. Urbán, J. Hartwig and I. Kocsis (2014), Combined Error Propagation Analysis and Runtime Event Detection in Process-Driven Systems, in International Workshop on Software Engineering for Resilient Systems, pp. 169–183, Springer

S. Andrews, G. Pruitt and B. Abernethy (2012), U.S. Department of Transportation: Signal Phase and Timing (Spat) Applications, Communications Requirements, Communications Technology Potential Solutions, Issues and Recommendations

M. Wallace (2005), Modular Architectural Representation and Analysis of Fault Propagation and Transformation, Electronic Notes in Theoretical Computer Science, vol. 141, no. 3, pp. 53–71, Springer

A. Wang, Y. Luo, G. Tu and P. Liu (2011), Vulnerability assessment scheme for power system transmission networks based on the fault chain theory, IEEE Transactions on power systems, 26, no. 1, 442–450, IEEE

J. Whittle, J. Hutchinson, and M. Rouncefield (2014), The State of Practice in Model-Driven Engineering, IEEE Software, vol. 31, no. 3, pp. 79–85, IEEE

I. Wolforth, M. Walker, Y. Papadopoulos and L. Grunske (2010), Capture and reuse of composable failure patterns, International Journal of Critical Computer-Based Systems, vol. 1, no. 1–3, pp. 128–147, Inderscience Publishers

A Nemzeti Közszolgálati Egyetem kiadványa

Kiadó:

Nemzeti Közszolgálati Egyetem
Közigazgatási Továbbképzési Intézet
www.uni-nke.hu

Felelős kiadó:
Prof. Dr. Kis Norbert rektorhelyettes
Címe: 1083 Budapest, Üllői út 82.

Kiadói szerkesztő:
Dorogi Katalin

Tördelőszerkesztő:
Friebert Máté

ISBN 978-963-498-397-2 (PDF)