24 KRITIKUS RENDSZEREK VALIDÁLÁSA

TÉMAKÖRÖK

A fejezet célja, hogy megtárgyalja a kritikus rendszerek fejlesztésénél használt verifikálási és validálási technikákat. A fejezet elolvasása után:

- megértjük, hogyan mérhető egy szoftverrendszer megbízhatósága, és hogyan használják a megbízhatóságnövekedési modelleket annak becslésére, hogy a kívánt megbízhatósági szintet mikor fogja elérni a rendszer;
- megismerjük a biztonságossági indoklások alapjait, és hogy hogyan lehet ezeket egyéb V & V módszerekkel együtt a rendszer biztonságosságának szavatolására használni;
- megismerkedünk a rendszer védettségének biztosításával kapcsolatos problémákkal;
- megértjük a biztonságossági indoklások alapgondolatát és azt, hogyan használhatók ezek annak demonstrálására, hogy a rendszerben nem következhet be veszélyhelyzet.

TARTALOM

Egy kritikus rendszer verifikálása és validálása nyilvánvalóan sok hasonlóságot mutat bármilyen más rendszer validálásával. A V & V folyamatoknak demonstrálniuk kell, hogy a rendszer megfelel a specifikációnak, továbbá a rendszer szolgáltatása és viselkedése támogatja a megrendelő követelményeit. Ugyanakkor a kritikus rendszerek esetén, ahol nagyfokú üzembiztosság szükséges, további tesztelési és elemzési lépésekre van szükség azért, hogy bizonyítékot szolgáltassanak a rendszer megbízhatóságára. Ennek két oka van:

1. *A hiba költsége.* A kritikus rendszerek hibáinak költsége és azok következményei sokkal nagyobbak lehetnek, mint a nemkritikus rendszereknél. Ha többet költünk a verifikálásra és a validálásra, akkor csökkentjük a rendszer meghibásodásának kockázatát. A hibákat általában olcsóbb még a rendszer átadása előtt megtalálni és kijavítani, mint a későbbi esetleges „balesetek" költségkövetkezményét vállalni.
2. *Az üzembiztonsági attribútumok validálása.* A kritikus rendszerek megrendelőinek meg kell győződniük arról, hogy a rendszer az előírt üzembiztonsági attribútumokat (rendelkezésre állás, megbízhatóság, biztonságosság és védettség) teljesíti. Ezeknek az attribútumoknak a megbecslése specifikus verifikálási és validálási tevékenységeket igényel, amelyekről a fejezet későbbi részében lesz szó. Bizonyos esetekben külső szabályzóknak – például a nemzeti légi közlekedési hatóságnak – kell igazolnia, hogy a rendszer biztonságos, még mielőtt telepítenénk. Ezt a bizonyítványt úgy kaphatjuk meg, ha speciális V & V eljárásokat dolgozunk ki és alkalmazunk, amelyek bizonyítékokat gyűjtenek a rendszer üzembiztosságára vonatkozóan.

Ezen okok miatt a V & V költsége kritikus rendszereknél általában sokkal magasabb, mint más rendszereknél. Kritikus szoftverrendszereknél nem szokatlan az, hogy a verifikálás és validálás a teljes fejlesztési költségnek több mint 50 százalékát felemészti. Ezt természetesen ellensúlyozza, ha így elkerülünk egy költséges rendszerhibát. Például 1996-ban az Ariane 5 rakétán lévő egyik kritikus szoftverrendszer kudarcot vallott, és több műhold megsemmisült. Az ebből eredő veszteség több százmillió dollár volt. A későbbi vizsgálat azt tárta fel, hogy a rendszer V & V folyamatában található hiányosságok részben felelősek a katasztrófáért.

Bár a kritikus rendszerek validálási folyamatának elsősorban a rendszer validálására kell helyeznie a hangsúlyt, verifikálni kell azt is, hogy meghatározott rendszerfejlesztési folyamatot használtak. Ahogyan azt a 27. és a 28. fejezetben kifejtem, a rendszer fejlesztéséhez használt folyamatok minősége kihatással van a rendszer minőségére. Röviden, a jó folyamatok jó rendszerhez vezetnek. Ezért olyan rendszerek fejlesztésénél, amelyekkel szemben nagy üzembiztonsági követelményeket támasztanak, bizonyosaknak kell lennünk abban, hogy megbízható fejlesztési folyamatot követünk.

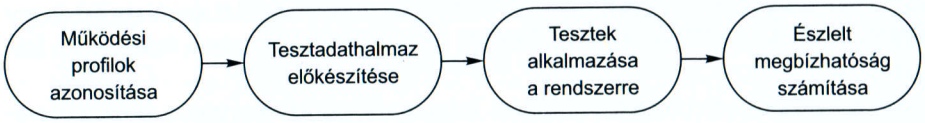
A folyamat szavatolása szerves része a minőségkezelésről szóló ISO 9000 szabványnak, amelyet a 27. fejezet röviden érint. Ez a szabvány megköveteli, hogy dokumentáljuk az alkalmazott folyamatokat és a hozzájuk kapcsolódó tevékenységeket, amelyek arról gondoskodnak, hogy kövessük a folyamatokat. Ez szokás szerint olyan kézzelfogható bizonyítékok generálását igényli, mint az aláírt űrlapok, mellyel igazolják a folyamattevékenységek és a termékminőség-ellenőrzések teljesítését. Az ISO 9000 szabvány előírja, milyen kézzelfogható folyamatkimeneteket kell előállítani, és ki a felelős ezeknek az űrlapoknak az előállításáért és ellenőrzéséért. A 24.2.2. alfejezetben ezt egy szabványos űrlap használatával szemléltetem, amely rögzíti a veszélyhelyzet-elemzési folyamatot.

24.1. MEGBÍZHATÓSÁG VALIDÁLÁSA

Ahogyan azt a 9. fejezetben kifejtettem, számos olyan metrikát fejlesztettek ki, amellyel egy rendszer megbízhatósági követelményei leírhatók. Annak validálásához, hogy egy rendszer kielégíti-e ezeket a követelményeket, úgy kell megmérnünk a rendszer megbízhatóságát, ahogyan azt egy tipikus felhasználó érzékeli.

Egy rendszer megbízhatósága mérésének folyamata látható a 24.1. ábrán. A folyamat a következő négy szakaszt foglalja magában:

1. Meglévő, azonos típusú rendszerek tanulmányozása, működési profil elkészítése céljából. A működési profilok azonosítják a rendszerbemenetek különböző osztályait és ezek valószínűségét szokásos használat esetén.
2. A működési profilt tükröző tesztsorozatok létrehozása. Ez azt jelenti, hogy tesztadatainkat a tanulmányozott rendszerek tesztadataival azonos valószínűség-eloszlással hozzuk létre. A folyamat támogatására tesztadat-generátorokat használhatunk.
3. A rendszer tesztelése ezekkel az adatokkal, és a hibák számának megfigyelése. A hibák száma naplózva is van. Ahogyan azt a 9. fejezetben kifejtettem, a használt megbízhatósági metrikának megfelelő időegységet javasolt választani.
4. Miután statisztikailag jelentős számú hibát figyeltünk meg, kiszámíthatjuk a szoftver megbízhatóságát. Ezután kidolgozhatjuk a megfelelő megbízhatósági metrikaértéket.



24.1. ábra. A megbízhatóság mérésének folyamata

Ezt a megközelítést gyakran *statisztikai tesztelés*nek hívják. A statisztikai tesztelés célja a rendszer megbízhatóságának felmérése. A hiányosságtesztelés célja ezzel szemben a rendszerhibák felderítése. Prowell és mások (Prowell *és mások*, 1999) a statisztikai tesztelést a Cleanroom-szoftvertervezésről szóló könyvükben tárgyalják.

A megbízhatóság mérésének ezt – az elméletét tekintve vonzó – megközelítését nem könnyű a gyakorlatban alkalmazni. Az alapvető nehézségek okai a következők:

1. *Bizonytalanságok a működési profilban.* Lehet, hogy a működési profil nem tükrözi pontosan a rendszer valós használatát, mivel más rendszerek tapasztalatai alapján alakítjuk ki.
2. *A tesztadatok generálásának magas költsége.* Nagy mennyiségű tesztadat generálása hosszú időt vesz igénybe, hacsak nem lehet a folyamatot teljességgel automatizálni.

3. *Statisztikai bizonytalanság nagy megbízhatóság előírása esetén.* Fontos, hogy statisztikailag jelentős számú hibát generáljunk ahhoz, hogy pontos megbízhatósági méréseket tegyünk lehetővé. Ha a szoftver már megbízható, akkor viszonylag kevés hiba történik, és nehéz újabb hibákat okozni.

Bizonyos típusú rendszereknél, melyeknek megszokott használati módjuk van (például telekommunikációs rendszerek), biztosan lehet pontos működési profilt fejleszteni. Más rendszerek esetén, ugyanakkor, számos különböző felhasználó létezik, akik a rendszert mind a saját módjukon használják. Ahogyan azt a 3. fejezetben leírtam, a különböző felhasználók egészen más benyomásokat szerezhetnek a megbízhatóságról, mivel különböző módokon használják a rendszert.

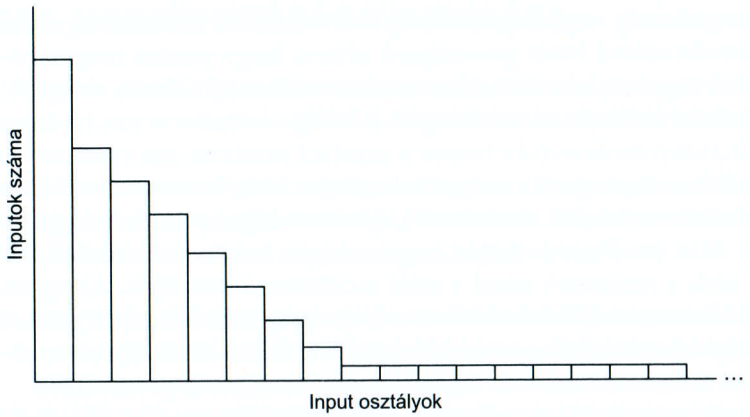
A megbízhatósági mérésekhez szükséges nagy adathalmaz generálásának messze a legjobb módja, ha valamilyen tesztadat-generátort használunk, amelyet be lehet állítani, hogy automatikusan a működési profilhoz illeszkedő bemeneteket állítson elő. Ugyanakkor interaktív rendszereknél általában nem lehet minden tesztadat előállítását automatikussá tenni. Ilyen rendszereknél az adathalmazokat kézileg kell generálni, ami ennek megfelelően magasabb költségekkel jár. Még ahol a teljes automatizálás lehetséges, ott is jelentős időt vesz igénybe a tesztadat-generátor számára történő utasítások megírása.

A rendszerek megbízhatóságának mérésekor a statisztikai bizonytalanság általános probléma. Ahhoz, hogy pontos megbízhatósági előrejelzést tegyünk, nem elég egyszerűen egyetlen rendszerhibát előidéznünk. Kellően nagyszámú hibát kell generálnunk ahhoz, hogy meggyőződjünk mérésünk pontosságáról. A probléma az, hogy minél jobbak vagyunk a rendszerbeli hibák számának minimalizálásában, annál nehezebbé válik a hibaminimalizálási technikák hatékonyságának mérése. Ha az előírt megbízhatóság nagyon magas szintű, gyakran nem praktikus e specifikációk ellenőrzéséhez elegendő rendszerhibát generálni.

24.1.1. Működési profilok

A szoftver működési profilja azt tükrözi, hogyan használják majd a gyakorlatban. A bemenet osztályainak specifikációjából és azok előfordulási gyakoriságából áll. Amikor egy meglévő manuális vagy automatizált rendszert új szoftverrendszerre cserélnek, meglehetősen egyszerű megbecsülni az új szoftver valószínű használati módját. Nagyjából meg fog egyezni a meglévő használati móddal, némi engedménnyel az új rendszerbe (feltételezhetően) belekerült új funkcionalitások tekintetében. Például egy működési profilt le lehet írni telekommunikációs kapcsolórendszereknél, mivel a telekommunikációs társaságok ismerik a rendszer által kezelendő hívási módokat.

A működési profil jellemzően olyan, hogy a legnagyobb valószínűséggel generált bemenetek kevés számú osztályba esnek, ahogy azt a 24.2. ábra bal oldala mutatja. Rendkívül magas azoknak az osztályoknak a száma, amelyeknél a bemenetek igen valószínűtlenek, de nem lehetetlenek. Ezt mutatja a 24.2. ábra jobb



24.2. ábra. Működési profil

oldala. A három pont (...) azt jelenti, hogy számtalan nem használt input van, ami már nem látható.

Musa (Musa, 1993; 1998) meghatározott irányelveket javasol a működési profilok fejlesztéséhez. Azon az alkalmazási szakterületen, amelyben ő dolgozott (telekommunikációs rendszerek), a használati adatok gyűjtése nagy múltra tekint vissza, így a működési profilok fejlesztésének folyamata viszonylag előrehaladott. Egy rendszernél, melynek fejlesztése kb. 15 emberévet vett igénybe, a működési profilt kb. egy emberhónap alatt fejlesztették ki. Más esetekben a működési profil generálása tovább tartott (2-3 emberév), de a költség természetesen megtérült a számos rendszerkiadás során. Musa kiszámolta, hogy a cégének (egy telekommunikációs társaságnak) a működési profil kifejlesztéséhez szükséges befektetése legalább tízszeresen megtérült.

Ugyanakkor, amikor a rendszer még új, nehezebb előre látni, hogyan fogják használni azt. A rendszereket számtalan felhasználó használja, különböző elvárásokkal, háttérrel és tapasztalatokkal. Nincs történeti használati adatbázis. A felhasználók gyakran olyan módon használják a rendszert, amire annak fejlesztői nem számítottak.

A problémát tovább bonyolítja az a tény, hogy a működési profil a rendszer használata során változhat. Ahogyan a felhasználók magabiztossága és képességei változnak, gyakorlatot szerezve a rendszerben, úgy kifinomultabb módon tudják majd azt használni. A nehézségek miatt Hamlet (Hamlet, 1992) azt mondja, hogy gyakran lehetetlen megbízható működési profilt szerkeszteni. Emiatt a megbízhatósági mérésekben nehéz megbecsülni a bizonytalanság fokát.

24.1.2. A megbízhatóság előrejelzése

A szoftver validálása során a vezetőknek a rendszer tesztelésére is energiát kell fordítaniuk. Mivel a tesztelés nagyon költséges, fontos, hogy ne „teszteljük túl" a rendszert. A tesztelést akkor lehet abbahagyni, amikor a kívánt szintű megbízha-

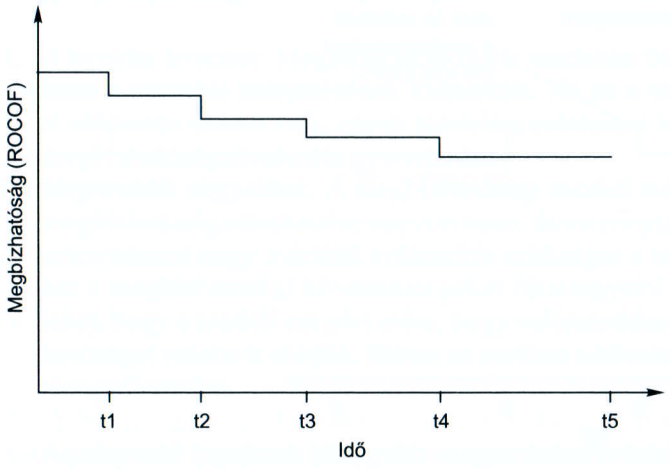
tóságot elértük. Természetesen előfordulhat, hogy a megbízhatósági előrejelzések azt mutatják, hogy a kívánt megbízhatósági szintet sohasem érjük el. Ebben az esetben a vezetőnek nehéz döntést kell meghoznia: vagy újraírják a szoftver egyes részeit, vagy újratárgyalják a szerződést.

A megbízhatóságnövekedési modell azt jellemzi, hogyan változik a rendszer megbízhatósága az időben a tesztelési folyamat során. Rendszerhibák megtalálásakor azok okait kijavítják, így a rendszer megbízhatóságának a tesztelés és nyomkövetés során javulnia kell. A megbízhatóság előrejelzéséhez a fogalmi megbízhatóságnövekedési modellt le kell fordítani matematikai modellre. Ennek részleteit nem taglalom, de a megbízhatóságnövekedés fogalmait röviden áttekintem.

Vannak különféle modellek, amelyek a számos különböző alkalmazási szakterületen szerzett megbízhatósági tapasztalatokból származnak. Kan (Kan, 2003) szerint a legtöbb modell exponenciális, és a hibák felfedezésével és kijavításával (24.5. ábra) a megbízhatóság gyorsan nő. Ez a növekmény azonban csökken, ahogyan egyre kevesebb hiányosság kerül napvilágra és törlésre a tesztelés későbbi szakaszaiban.

A legegyszerűbb megbízhatóságnövekedési modell egy lépcsősfüggvény-modell (Jelinski és Moranda, 1972), ahol a megbízhatóság mindig egy konstans növekménnyel növekszik, amikor egy hibát felfedeznek és kijavítanak (24.3. ábra). Ez a modell feltételezi azt, hogy a javításokat a szoftverben mindig helyesen viszik véghez, így a szoftverhibák és a hozzájuk kapcsolódó sikertelenségek száma idővel csökken. A javítások elvégzése után a szoftverhibák előfordulási gyakoriságának (rate of occurrence of software failures, ROCOF) ezért csökkennie kell, ahogyan az a 24.3. ábrán látható. Figyeljük meg, hogy a vízszintes tengely mentén lévő időtartamok a rendszer tesztelésre bocsátásai között eltelt időket tükrözik, ezért, természetes módon, nem egyenlő hosszúak.

A gyakorlatban azonban a nyomkövetéssel nem mindig sikerül helyrehozni a szoftverhibákat, és a kód megváltozása új hibákat okozhat a rendszerben.



24.3. ábra. A megbízhatóságnövekedés egyenletes lépcsős függvényű modellje

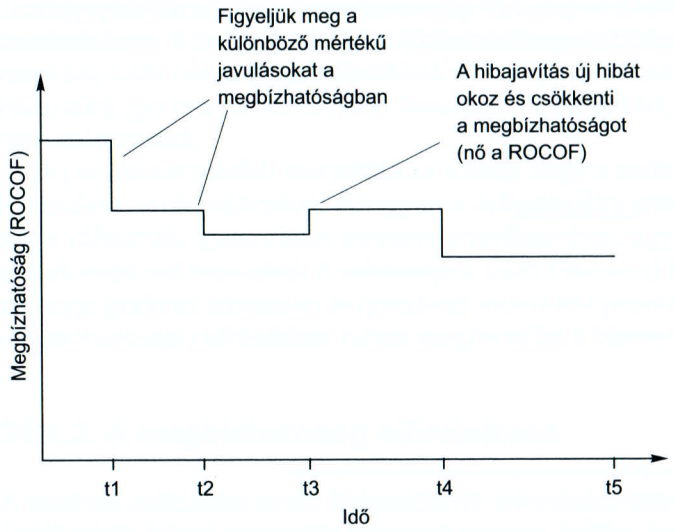
E hibák előfordulási gyakorisága nagyobb lehet az éppen kijavított hiba előfordulási valószínűségénél. Emiatt a rendszer megbízhatósága akár rosszabbá is válhat ahelyett, hogy javulna.

Az egyszerű, egyenletes lépcsős megbízhatóságnövekedési modell azt is feltételezi, hogy a hibák egyenlő mértékben járulnak hozzá a megbízhatósághoz, és minden hibajavítás is ugyanolyan mértékben növeli a megbízhatóságot. Azonban nem minden hiba egyformán valószínű. A leggyakoribb hibák kijavítása jobban hozzájárul a megbízhatóság növekedéséhez, mint az alkalmanként előforduló hibáké. Ezeket a lehetséges hibákat valószínűleg korán, még a tesztelési folyamat elején megtalálhatjuk, így a megbízhatóság jobban növelhető, mint később, amikor a kevéssé valószínű hibákat derítjük fel.

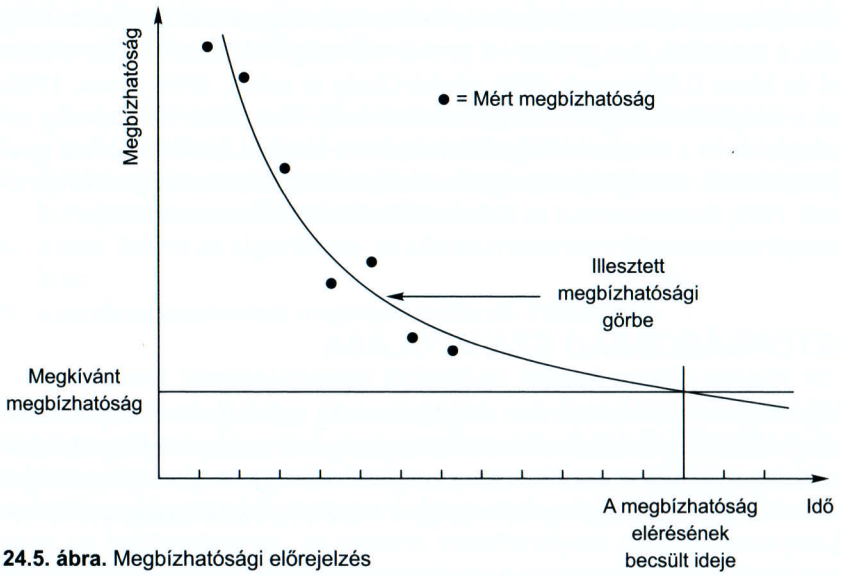
Későbbi modellek, mint például a Littlewood és Verrall (Littlewood és Verrall, 1973) által javasolt modell, ezeket a problémákat úgy kezelik, hogy egy véletlenszerű elemet vezetnek be a szoftverjavítással elért megbízhatóságnövekedés javulásába. Ezért nem minden javítás eredményez ugyanolyan mértékű javulást a megbízhatóságban, hanem az egy véletlenszerű perturbációtól függően változik (24.4. ábra).

Littlewood és Verrall modellje megengedi a megbízhatóság negatív növekedését, ha a szoftverjavítás további hibákat okoz. Szintén modellezi azt a tényt, hogy hibák kijavításakor a megbízhatóságban javításonként bekövetkezett átlagos javulás csökken. Ennek az oka az, hogy a legvalószínűbb hibákat valószínűleg a tesztelési folyamat korai szakaszában megtaláljuk. Ezek kijavítása járul hozzá legnagyobb mértékben a megbízhatóság növekedéséhez.

A fentiek diszkrét modellek, amelyek inkrementális megbízhatóságnövekedést tükröznek. Ha a szoftver egy új verzióját átadják tesztelésre, kijavított hibákkal, annak alacsonyabb hiba-előfordulási rátával kell rendelkeznie, mint a ko-



24.4. ábra. A megbízhatóságnövekedés véletlen ugrású lépcsős függvényű modellje



24.5. ábra. Megbízhatósági előrejelzés

rábbi verziónak. Ugyanakkor ahhoz, hogy bizonyos mennyiségű tesztelés után elért megbízhatóságot előre jelezzünk, folytonos matematikai modellekre van szükség. Különböző alkalmazási szakterületekről származó számos különböző modellt mutattak be és hasonlítottak össze (Littlewood, 1990).

Nagyon leegyszerűsítve, a megbízhatóságot úgy tudjuk előre jelezni, hogy a mért megbízhatósági adatokat összevetjük egy ismert megbízhatósági modellel. Ezt a modellt azután extrapoláljuk a megbízhatóság kívánt szintjére. Az időpont, amikor ezt elérjük, ezután leolvasható a grafikonról (24.5. ábra). A tesztelést és nyomkövetést tehát eddig az időpontig kell folytatni.

Két előnye van annak, ha a rendszer megbízhatóságát megbízhatósági modell alapján jósoljuk meg:

1. *A tesztelés tervezése.* Megadva az aktuális tesztelési ütemtervet, előre lehet jelezni a tesztelés befejezésének időpontját. Ha ez a rendszer tervezett átadási időpontja utánra esik, egyéb tesztelési erőforrást kell hadrendbe állítani a megbízhatóságnövekedés gyorsítására.
2. *Megrendelői tárgyalások.* A megbízhatósági modell néha azt mutatja, hogy a megbízhatóság növekedése nagyon lassú, és viszonylag kis haszon eléréséhez aránytalanul nagy mértékű erőfeszítés szükséges a tesztelésben. Érdemes lehet a megbízhatósági követelményeket újratárgyalni a megrendelővel. Az is lehet, hogy a modell azt jelzi előre, hogy valószínűtlen, hogy a kívánt megbízhatóságot valaha is elérjük. Ebben az esetben a követelmények újratárgyalása elengedhetetlen.

Az alapvető fogalmak könnyebb magyarázása érdekében leegyszerűsítettem a megbízhatóságnövekedési modellt. Azoknak az olvasóknak, akik ezt ténylege-

sen alkalmazni szeretnék, mélyebben bele kell ásniuk magukat a témába, és meg kell érteniük a modellek és a gyakorlati problémái mögött húzódó matematikát. Littlewood és Musa (Littlewood, 1990; Abdel-Ghaly *és mások*, 1986; Musa, 1998) bőven írtak a megbízhatóságnövekedési modellekről, Kan (Kan, 2003) pedig remek összefoglalót írt a témáról könyvében. Számos további szerző is leírta gyakorlati tapasztalatait a megbízhatóságnövekedési modellek használatáról (Ehrlich *és mások*, 1993; Schneidewind és Keller, 1992; Sheldon *és mások*, 1992).

24.2. BIZTONSÁGOSSÁG SZAVATOLÁSA

A biztonságosság szavatolásának és a megbízhatóság validálásának célja különbözik. A megbízhatóság valamilyen metrika segítségével mennyiségileg előírható, azután pedig mérhető a kész rendszer megbízhatósága. A biztonságosságot azonban nem lehet mennyiségileg előírni, így a rendszer tesztelésekor sem lehet mérni azt.

A biztonságosság szavatolása ezért egy bizalmi szintet próbál megállapítani a rendszerben, ami a „nagyon alacsonytól" a „nagyon magasig" változhat. Ennek elbírálása szakértők dolga, és a rendszerről, a környezetéről és fejlesztési folyamatairól szóló bizonyítékokra épít. Ez a bizalmi szint számos esetben részben a rendszert fejlesztő szervezet tapasztalatán alapszik. Ha egy vállalat előzőleg számos vezérlőrendszert fejlesztett, amelyek biztonságosan működtek, akkor észszerű feltételezni, hogy ezentúl is biztonságos rendszereket fog fejleszteni ebből a típusból.

Ugyanakkor egy ilyen értékelést alá kell támasztani a rendszertervből, a rendszer verifikálásának és validálásának eredményéből és a használt rendszerfejlesztési folyamatból származó kézzelfogható bizonyítékokkal. Bizonyos rendszerek esetében ezek a kézzelfogható bizonyítékok egy biztonságossági esetbe (24.4. alfejezet) vannak gyűjtve, ami lehetőséget ad arra, hogy egy külső szereplő is arra a következtetésre juthasson, hogy a fejlesztőrendszer biztonságosságába vetett bizalmi szintje helytálló.

A biztonságosságkritikus rendszerek verifikálása és validálása sok hasonlóságot mutat más, nagy üzembiztonsági követelményekkel rendelkező rendszerek tesztelésével. Széles körű tesztelést kell folytatni, hogy annyi hibát felfedezzünk, ahányat csak lehetséges, és ajánlott statisztikai tesztelést alkalmazni a rendszer megbízhatóságának megbecsléséhez. A számos rendszerben megkövetelt nagyon alacsony meghibásodási ráta miatt azonban a statisztikai tesztelés nem tud mindig mennyiségi becslést adni a rendszer megbízhatóságára. A tesztek a rendszer biztonságosságának elbírálásához egyszerűen olyan bizonyítékokat adnak, amiket más bizonyítékokkal együtt lehet felhasználni. Ilyenek például a felülvizsgálatok eredményei és a statikus ellenőrzés (22. fejezet).

A széles körű felülvizsgálat nélkülözhetetlen a biztonságosság-központú fejlesztési folyamat során. Olyan embereknek kell megmutatni a szoftvert, akik különböző nézőpontból tekintenek majd rá. Parnas és mások (Parnas *és mások*,

1990) ötfajta felülvizsgálatot javasolnak, amelyeket biztonságosságkritikus rendszereknél kötelezőként írna elő:

1. a funkciók eredeti szándék szerinti megvalósulásának vizsgálata;
2. a karbantartható, érthető szerkezet vizsgálata;
3. felülvizsgálat annak verifikálására, hogy az algoritmus- és adatszerkezetterv konzisztens-e az előírt viselkedéssel;
4. a kód, illetve az algoritmus- és adatszerkezetterv konzisztenciájának vizsgálata;
5. a rendszertesztesetek megfelelő voltának áttekintése.

A rendszer biztonságossága érdekében végzett munka mögött az a feltételezés húzódik meg, hogy az esetlegesen veszélyt okozó rendszerhibák száma lényegesen kevesebb, mint a rendszerben létező tényleges hibák teljes száma. A biztonságosság szavatolása ezekre a potenciálisan veszélyt okozó hibákra koncentrálhat. Ha meg lehet mutatni, hogy ezek a hibák nem következhetnek be, vagy ha bekövetkeznek, az okozott veszélyhelyzet nem eredményez balesetet, akkor a rendszer biztonságos. Ez az alapja a biztonságosság indoklásának, ami a következő szakasz tárgya.

24.2.1. Biztonságossági indoklások

A 22. fejezetben leírt programhelyesség-bizonyítás több mint 30 éve van jelen a szoftververifikálási technikák között. A kis rendszerek számára biztosan el lehetne készíteni formális programbizonyításokat. A helyesség bizonyításának gyakorlati problémái olyan nagyok, hogy kevés szervezet találta őket költséghatékonynak a szokásos rendszerfejlesztési folyamatok során. Bizonyos kritikus alkalmazások esetén azonban gazdaságos lehet a helyesség bizonyítása azért, hogy növeljük a bizalmat az iránt, hogy a rendszer kielégíti a vele szemben támasztott biztonságossági és védettségi követelményeket. Ez különösen fontos abban az esetben, amikor a biztonságosságkritikus funkcionalitás jól körülhatárolhatóan egy viszonylag kis, formálisan megadható alrendszerbe vihető.

Bár lehet, hogy a legtöbb rendszernél nem költséghatékony helyességbizonyításokat készíteni, néha lehetséges olyan biztonságossági indoklást találni, amely megmutatja, hogy a program teljesíti biztonságossági kötelezettségeit. Nem szükséges bizonyítani azt, hogy a program megfelel a specifikációjának. Csak azt szükséges megmutatni, hogy a program végrehajtása nem vezethethet nem biztonságos állapothoz.

A rendszer biztonságossága demonstrálásának leghatékonyabb technikája az ellentmondáson alapuló bizonyítás. Ez azt jelenti, hogy feltételezzük, hogy a program végrehajtásával el lehet érni egy (a veszélyhelyzet-elemzés által azonosított) nem biztonságos állapotot. Ezután szisztematikusan elemezzük a kódot, és megmutatjuk, hogy a veszélyes állapot előfeltételei ellentmondanak az ahhoz az állapothoz vezető programágak utófeltételeinek. Ha ez fennáll, a nem bizton-

```
– A beadott inzulinadag a vércukorszint, az előzőleg beadott adag és az
– előző beadás idejének függvénye

    currentDose = computeInsulin () ;
    // Biztonságossági ellenőrzés – a currentDose beállítása szükség esetén
    // if utasítás 1
    if (previousDose == 0)
    {
        if (currentDose > 16)
            currentDose = 16 ;
    }
    else
        if (currentDose > (previousDose * 2) )
            currentDose = previousDose * 2 ;
    // if utasítás 2
    if ( currentDose < minimumDose )
            currentDose = 0 ;
    else if ( currentDose > maxDose )
            currentDose = maxDose ;
    administerInsulin (currentDose) ;
```

24.6. ábra. Az inzulinadagolás kódja

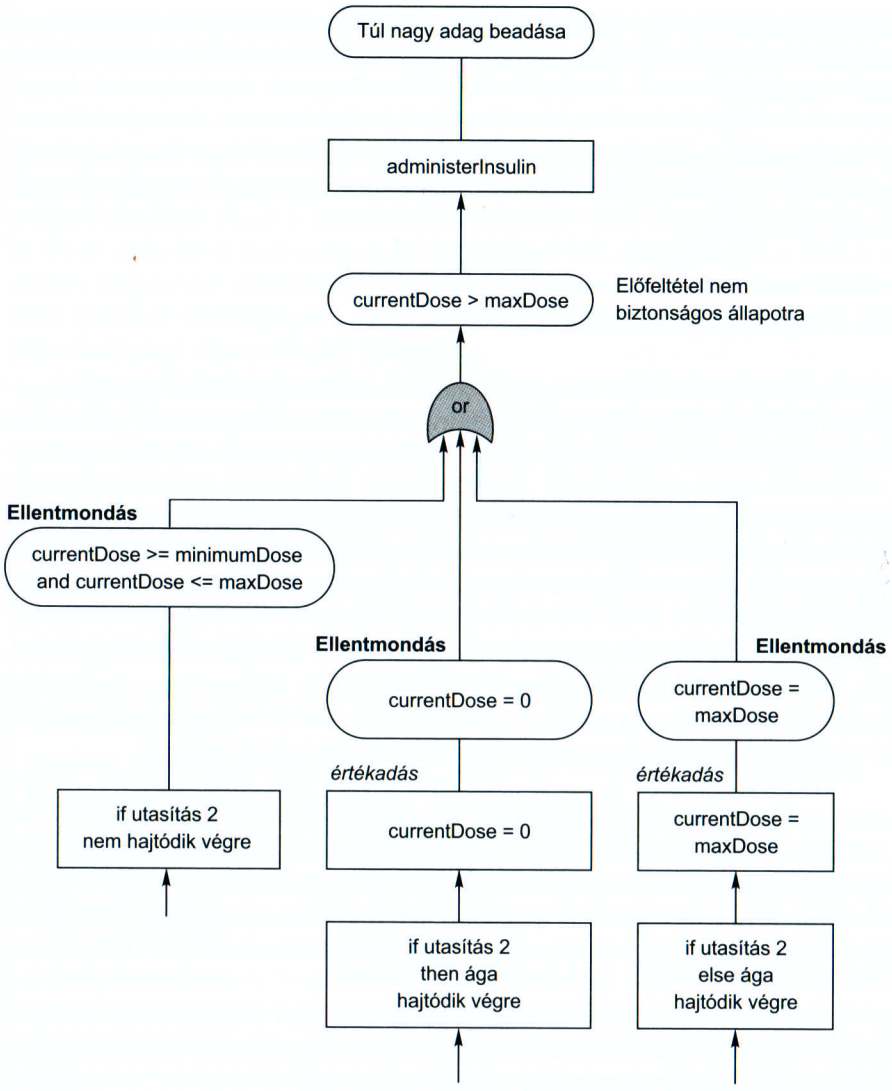
ságosságra vonatkozó kezdeti feltevésünk hamis volt. Ha ezt minden azonosított veszélyhelyzetre elvégezzük, akkor a szoftver biztonságos.

Példaként tekintsük a 24.6. ábrán szereplő kódot, amely az inzulinadagoló rendszer megvalósításának része lehet. A biztonságossági indoklás elkészítéséhez meg kell mutatni, hogy a beadott inzulinadag soha nem nagyobb egy bizonyos maximális szintnél, amelyet minden betegnél egyénileg határoznak meg. Tehát nem szükséges bizonyítani, hogy a rendszer a „helyes" adagot adja, elég csupán azt, hogy sohasem adagolja túl az inzulint.

A biztonságosság indoklásához azonosítjuk a nem biztonságos állapot előfeltételét, ami ebben az esetben currentDose > maxDose. Aztán megmutatjuk, hogy minden programág ellentmondáshoz vezet ezzel az állítással. Ebben az esetben a nem biztonságosság feltétele nem lehet igaz. Tehát a rendszer biztonságos. A biztonságossági indoklások grafikus megjelenítése a 24.7. ábrán látható.

A biztonságossági indoklások – mint amilyen a 24.7. ábrán látható – sokkal rövidebb ideig tartanak, mint a formális rendszerverifikációk. Először azonosítunk minden lehetséges programágat, amely a potenciálisan nem biztonságos állapothoz vezet. Ettől a nem biztonságos állapottól kezdve visszafelé dolgozunk, és minden állapotváltozónál a nem biztonságos állapothoz vezető ágakon az utolsó értékadást tekintjük. Az előző számításokat (például a 24.7. ábrán szereplő if utasítás 1-et) nem kell figyelembe venni a biztonságossági indoklásban. Ennél a példánál a currentDose lehetséges értékeire kell tekintettel lennünk közvetlenül az administerInsulin módszer végrehajtása előtt.

A 24.7. ábrán látható biztonságossági indoklásban három lehetséges programág vezethet az administerInsulin módszer hívásához. Azt kívánjuk demonstrálni,



24.7. ábra. Ellentmondás felmutatásán alapuló informális biztonságossági indoklás

hogy az inzulinadagolás mértéke soha nem haladja meg a maxDose értéket. Az administerInsulin-hoz vezető programágak:

1. Az if utasítás 2 egyik ága sem hajtódik végre. Ez csak akkor következhet be, ha a currentDose nagyobb vagy egyenlő a minimumDose-nál, és kisebb vagy egyenlő a maxDose-nál.
2. Az if utasítás 2 then ága hajtódik végre. Ebben az esetben a currentDose-hoz nullát rendelő értékadás hajtódik végre. Tehát ennek utófeltétele currentDose = 0.

3. Az if utasítás 2 else ága hajtódik végre. Ebben az esetben a currentDose-hoz maxDose-t rendelő értékadás hajtódik végre. Tehát ennek utófeltétele currentDose = maxDose.

Az utófeltételek mindhárom esetben ellentmondanak a nem biztonságosság előfeltételének (vagyis hogy az adagolás nagyobb lenne, mint a maxDose), tehát a rendszer biztonságos.

24.2.2. Folyamat szavatolása

A fejezet bevezetésében már hangsúlyoztam a rendszerfejlesztési folyamat minősége szavatolásának fontosságát. Ez minden kritikus rendszernél fontos, de biztonságosságkritikus rendszerek esetén különösen az. Ennek két oka van:

1. A balesetek ritka események a kritikus rendszerekben, és gyakorlatilag lehetetlen szimulálni őket egy rendszer tesztelése során. Nem számíthatunk arra, hogy akár az átfogó tesztelés is meg tudná ismételni a balesethez vezető feltételeket.
2. A biztonságossági követelmények, ahogyan azt a 9. fejezetben tárgyaltam, néha „nem szabad" követelmények, melyek a nem biztonságos viselkedést zárják ki. Lehetetlen meggyőzően demonstrálni teszteléssel vagy más validálási tevékenységekkel, hogy ezek a követelmények teljesítve lettek.

A biztonságosságkritikus rendszerek fejlesztésének életciklusmodellje (9. fejezet, 9.7. ábra) világossá teszi, hogy a szoftverfolyamat minden szakaszában határozott figyelmet kell fordítani a biztonságosságra. Ez azt jelenti, hogy a folyamatnak olyan tevékenységeket is magában kell foglalnia, amelyek szavatolják a specifikus biztonságosságot. Ezek a következőket tartalmazzák:

1. Egy veszélyhelyzet-naplózó és -monitorozó rendszer létrehozását, amely a veszélyhelyzeteket az előkészítő veszélyhelyzet-elemzéstől a tesztelésig és a rendszer validálásáig nyomon követi.
2. A projekt biztonságossági mérnökeinek kijelölését, akik közvetlenül felelősek a rendszer biztonságossági szempontjaiért.
3. A biztonsági felülvizsgálatok kiterjedt használatát a fejlesztési folyamat egészében.
4. Biztonságosság minősítési rendszerének létrehozását, amivel a biztonságosságkritikus komponensek becsült biztonságosságát hivatalosan minősítik.
5. Nagyon részletes konfigurációkezelési rendszer használatát (29. fejezet), amellyel nyomon követhető az összes biztonságossággal kapcsolatos dokumentum, és szinkronban tartható a kapcsolódó technikai dokumentációval. Kevés érv szól amellett, hogy szigorú validálási eljárásokat használjunk, ha a konfigurációkezelésben bekövetkezett egyetlen hiba azt jelenti, hogy rossz rendszert adtak át a megrendelőnek.

Annak szemléltetésére, hogy mit foglal magában a folyamatvalidálás, a vészhelyzetelemzés folyamatát használjuk, amely a biztonságosságkritikus rendszerek fejlesztésének elengedhetetlen részét képezi. A veszélyhelyzet-elemzés a veszélyhelyzetek azonosításával, bekövetkezési valószínűségükkel és annak valószínűségével foglalkozik, hogy balesethez vezetnek. Ha a fejlesztési folyamat nyomon követhető a veszélyhelyzetek azonosításától magáig a rendszerig, akkor megindokolható, hogy a veszélyhelyzetek miért nem okoznak balesetet. Ezeket ki lehet egészíteni biztonságossági indoklásokkal, ahogyan azt a 24.2.1. alfejezetben tárgyaltam. Ahol külső minősítés szükséges a rendszer használatbavétele előtt (például repülőgépen), ott a minősítés szokásos feltétele, hogy a nyomon követhetőséget demonstrálni lehessen.

A központi biztonságossági dokumentum a veszélyhelyzetnapló, ahol a specifikációs folyamat során azonosított veszélyhelyzetek vannak dokumentálva és nyomon követve. Ezt a veszélyhelyzetnaplót a szoftverfejlesztési folyamat minden állomásában használják ezután annak felmérésére, hogy mennyire veszi figyelembe az adott szakasz a veszélyhelyzeteket. A 24.8. ábrán egyszerűsített példa látható az inzulinadagoló rendszer veszélyhelyzetnaplójának egy bejegyzésére. Ez dokumentálja a veszélyhelyzet-elemzés folyamatát, és bemutatja a folyamat során generált tervezési követelményeket. Ezek a követelmények azt hivatottak biztosítani, hogy a vezérlőrendszer sosem adagolja túl az inzulint.

Veszélyhelyzetnapló			**4. lap: nyomtatva 2003. 02. 20.**
Rendszer: Inzulinpumpa-rendszer		*Állomány:* InzulinPump/Safety/HazardLog	
Biztonságossági mérnök: James Brown		*Napló verzió:* 1/3	
Azonosított veszélyhelyzet	Inzulin túladagolása a betegnek		
Azonosította	Jane Williams		
Kritikussági osztály	1		
Azonosított kockázat	Magas		
A hibafa azonosítva	IGEN	*Dátum* 99.01.24.	*Helye* Veszélyhelyzetnapló 5. o.
A hibafa létrehozói	Jane Williams és Bill Smith		
A hibafa ellenőrizve	IGEN	*Dátum* 99.01.28.	*Ellenőrizte* James Brown

Rendszer-biztonságosság tervezési követelmények

1. A rendszernek öntesztelő szoftvert kell tartalmaznia, amely teszteli majd az érzékelő szoftvert, az órát és az inzulinadagoló rendszert.
2. Az önellenőrző szoftvernek percenként egyszer végre kell hajtódnia.
3. Abban az esetben, ha az önellenőrző szoftver bármelyik rendszerkomponensben hibát észlel, hallható figyelmeztetést kell kiadni, és a pumpa kijelzőjének jeleznie kell azt a komponenst, amelyben a hibára fény derült. Az inzulin szállítását javallott felfüggeszteni.
4. A rendszernek magában kell foglalnia egy felülbíráló rendszert, amely lehetővé teszi a használónak, hogy módosítsa a rendszer által beadandó inzulin kiszámított adagját.
5. A felülbírálás mennyiségének javallott korlátozva lennie, hogy ne lehessen nagyobb egy előre beállított értéknél, ami akkor kerül beállításra, amikor a rendszert az orvosi személyzet bekonfigurálja.

24.8. ábra. Egyszerűsített veszélyhelyzetnapló oldala

Ahogyan az a 24.8. ábrán látható, mindenképpen érdemes azonosítani azokat a személyeket, akik bizonyos biztonságossági követelményekért felelősek. Fontos kijelölni egy projektbiztonságossági mérnököt, akinek a rendszer fejlesztésében nem szabad részt vennie. Ennek a mérnöknek a felelőssége biztosítani, hogy a megfelelő biztonsági ellenőrzéseket elvégezték és dokumentálták. A rendszernek szüksége lehet egy külső szervezettől érkezett független biztonságossági felügyelőre, aki a biztonságossági dolgokról közvetlenül az ügyfélnek tesz jelentést.

A biztonságossági felelősséggel rendelkező rendszermérnököknek számos alkalmazástípusnál minősítetteknek kell lenniük. Az Egyesült Királyság területén ez azt jelenti, hogy egy mérnöki intézetnek (általános, villamos, gépész stb.) tagjává kell fogadnia, és okleveles mérnöknek kell lennie. Tapasztalatlan, gyengén képzett mérnökök nem vállalhatnak felelősséget a biztonságosságért.

Ez jelenleg a szoftvermérnökökre nem vonatkozik, habár széles körű párbeszéd zajlik az Egyesült Államok több államában is a szoftvermérnökök engedélyhez kötéséről (Knight és Leveson, 2002; Bagert, 2002). A biztonságosságkritikus szoftvertervezés jövőbeli folyamatszabványai megkövetelhetik, hogy a projekt biztonsági mérnökei meghatározott minimális képzési szintet teljesített, hivatalosan minősített mérnökök legyenek.

24.2.3. Futási idejű biztonságosság-ellenőrzés

A 20. fejezetben vezettem be a defenzív programozás fogalmát, amelyben redundáns utasításokat adtunk a programokhoz azért, hogy ellenőrizzük a lehetséges rendszerhibákat. Hasonló technika alkalmazható a biztonságosságkritikus rendszerek dinamikus figyeléséhez is. Ellenőrző kódot lehet adni a programhoz, amely ellenőriz egy biztonságossági megszorítást, és kivételt vált ki, ha ez a megszorítás sérül. A program bizonyos pontjain érvényes biztonságossági megszorításokat *előírások* segítségével fejezhetjük ki. Az előírások olyan feltételeket leíró predikátumok, amelyeknek teljesülniük kell ahhoz, hogy a következő utasítást végre lehessen hajtani. Biztonságosságkritikus rendszereknél az előírásokat a biztonságossági specifikáció alapján érdemes generálni. Ezek a biztonságos viselkedés szavatolására szolgálnak, nem pedig a specifikációnak történő megfelelést tűzik ki célul.

Az előírások különösen értékesek lehetnek a rendszer komponensei közötti kommunikáció biztonságosságának szavatolásában. Például az inzulinadagoló rendszer esetén az inzulinadag bejuttatása úgy történik, hogy jelzések generálódnak az inzulinpumpának, hogy előírt számú inzulinegységet szállítson (24.9. ábra). A megengedett maximális inzulinadaghoz tartozó inzulinegységek száma előre kiszámítható, és előírás formájában a rendszerbe illeszthető.

Ezért, ha hiba történt a currentDose kiszámításában (ez az állapotváltozó tárolja a szállítandó inzulin mennyiségét), vagy ha ez az érték valamilyen módon megsérül, az ebben a szakaszban el lesz fogva. Túlzott inzulinadag nem kerülhet szállításra, mivel a metódusban lévő ellenőrzés biztosítja, hogy a pumpa nem szállíthat a maxDose-nál nagyobb adagot.

```
static void administerInsulin () throws SafetyException {
        int maxIncrements = InsulinPump.maxDose / 8 ;
        int increments = InsulinPump.currentDose / 8 ;
        // assert currentDose <= InsulinPump.maxDose
        if (InsulinPump.currentDose > InsulinPump.maxDose)
                throw new SafetyException (Pump.doseHigh);
        else
                for (int i = 1; i <= increments; i++)
                {
                        generateSignal () ;
                        if (i > maxIncrements)
                            throw new SafetyException(Pump.incorrectIncrements);
                }// for ciklus
    } // administerInsulin
```

24.9. ábra. Inzulinadagolás adminisztrációja futási idejű ellenőrzéssel

A megjegyzések formájában befoglalt biztonságossági előírásokból ezeket az előírásokat ellenőrző kód generálható, amint azt a 24.9. ábra mutatja. Elvben ezen a kód generálásának nagy részét automatizálni lehetne egy előírás-előfeldolgozó segítségével. Ezek azonban speciális eszközök, ezért az előírások kódját rendszerint kézzel írják.

24.3. VÉDETTSÉG ÉRTÉKELÉSE

Ahogy egyre több rendszert kötnek az internetre, és válik elérhetővé egy hálózati kapcsolat segítségével, úgy a rendszer védettségének értékelése is egyre fontosabbá válik. Naponta hallani történeteket webalapú rendszerek elleni támadásokról, illetve vírusokról és férgekről, amelyek az internetprotokollok használatával próbálnak elterjedni.

Mindez azt jelenti, hogy a webalapú rendszerek verifikációs és validációs folyamatainak a védettség értékelését kell a középpontba helyezniük, ahol a rendszer különféle támadásokkal szembeni ellenálló képességét teszteljük. Anderson szerint azonban (Anderson, 2001) ezt a fajta védettségi értékelést nagyon nehéz elvégezni. Ennek következményeként a rendszerek gyakran olyan biztonsági résekkel együtt kerülnek telepítésre, amelyek hozzáférést, vagy szélsőséges esetben a rendszer lerombolását engedik a támadónak.

Alapvetően annak oka, hogy miért olyan nehéz a védettséget kiértékelni, az, hogy a védettségi követelmények hasonlóak a biztonságossági követelményekhez, vagyis ezek „nem szabad" követelmények, azaz ezek a rendszer számára tiltott, nem pedig a megengedett viselkedést szabják meg. Azonban gyakran nincs mód ezt a viselkedést egyszerű, a rendszer által ellenőrizhető megszorítások formájában megadni.

Megfelelő erőforrások birtokában mindig meg lehet mutatni, hogy egy rendszer megfelel-e funkcionális követelményeinek, azonban lehetetlen bebizonyíta-

ni, hogy nem csinál valamit, magyarán a tesztelés mennyiségétől függetlenül biztonságilag sebezhető rész biztosan marad majd a telepítés után is. Egy leleményes támadó még egy évek óta használatban lévő rendszerben is felfedezhet új támadási formákat, és behatolhat egy korábban védettnek gondolt rendszerbe. Például a biztonságosnak gondolt RSA adattitkosítási algoritmust 1999-ben feltörték.

A védettség ellenőrzésének négy egymást kiegészítő megközelítése van:

1. *Tapasztalatalapú validálás.* Ebben az esetben a rendszert a validálócsoport által ismert támadási típusok elleni védelem szempontjából elemzik. Ezt a típusú validálást általában az eszközalapú validálással együtt alkalmazzák. Az ismert biztonsági problémákról érdemes egy ellenőrző listát készíteni (24.10. ábra). Ekkor az összes rendszer-dokumentáció használható.
2. *Eszközalapú validálás.* Ebben az esetben a rendszer elemzésére olyan változatos védelmi eszközöket használhatnak, mint például a jelszóellenőrzők. Ez valójában egy kiegészítés a tapasztalatalapú validáláshoz, ahol a tapasztalatot a használt eszközök testesítik meg.
3. *Tigris csoportok.* Ebben az esetben egy csoportot állítanak fel, amelynek feladata a rendszer védelmének feltörése. Támadásokat szimulálnak a rendszeren, és ügyességüket arra használják, hogy a rendszervédelem veszélyeztetésének új módjait fedezzék fel. Ez nagyon hatékony módszer, különösen akkor, ha a csapat tagjai rendelkeznek már korábbi betörési tapasztalatokkal.
4. *Formális verifikálás.* A rendszert verifikálni lehet egy formális védelmi specifikáció szerint. Ugyanakkor, ahogy más területeken, a formális verifikálást a védelem területén sem használják széles körben.

Egy rendszer végfelhasználói számára a védettség verifikálása igen nehéz. Következésképpen, amint azt Gollmann (Gollmann, 1999) írta, európai és északamerikai testületek olyan védelemértékelési kritériumokat fektettek le, amiket specializált értékelők ellenőrizni tudnak. A szoftvertermékek készítői a termékeiket elküldhetik értékelésre, és e kritériumok alapján minősíttethetik azokat.

Biztonsági ellenőrző lista

1. Minden, az alkalmazás által létrehozott állomány megfelelő hozzáférési jogosultságokkal bír? Rossz jogosultságok az állományok jogtalan felhasználók által történő hozzáférését eredményezhetik.
2. A rendszer egy idő után automatikusan befejezi a felhasználói munkameneteket? Egy felügyelet nélküli gépen aktívan hagyott munkamenet illetéktelen hozzáférést biztosíthat a rendszerhez.
3. Ha a rendszer megvalósítása olyan programozási nyelven történt, amely nem ellenőrzi a tömbhatárokat, akkor előfordulhat-e, hogy túlcsordulás következik be? Ez ugyanis lehetővé teszi a támadóknak, hogy végrehajtható kódot küldjenek és hajtsanak végre.
4. Ha jelszavakat használunk azonosításra, ellenőrzi a rendszer, hogy ezek a jelszavak erősek-e. Az erős jelszavak vegyesen tartalmaznak betűket, számjegyeket és speciális karaktereket is, és nem szótári szavak. Ezeket sokkal nehezebb feltörni, mint az egyszerű jelszavakat.

24.10. ábra. Egy biztonsági ellenőrző lista bejegyzései

Ha a követelmények bizonyos szintű védelmet írnak elő, választhatunk olyan terméket, amit arra a szintre minősítettek. Ugyanakkor számos termék nem rendelkezik védelmi minősítéssel, és a minősítés egyedi termékekre vonatkozik. Ha a minősített rendszert más, nem minősített rendszerekkel együtt használják, például helyi fejlesztésű szoftverekkel, akkor a teljes rendszer védelmi szintje nem megbecsülhető.

24.4. BIZTONSÁGOSSÁGI ÉS ÜZEMBIZTONSÁGI ESETEK

A biztonságossági – vagy még általánosabban: üzembiztonsági – esetek olyan strukturált dokumentumok, amelyek részletes indoklásokat és bizonyítékokat sorolnak fel arra vonatkozóan, hogy a rendszer biztonságos és eléri a szükséges üzembiztonsági szintet. A kritikus rendszerek sok fajtája esetén a biztonságossági eset elkészítése valós követelmény, és ennek a rendszer telepítését megelőzően eleget kell tennie egy tanúsítványnak.

A kormányok különböző szabályzókat alkotnak annak érdekében, hogy az ipari szereplők ne kerülhessék meg a nemzeti biztonságossági, védelmi és egyéb szabványokat. Ilyen szabályzók több iparágban is jelen vannak, például a légitársaságokra a nemzeti légi közlekedési hivatalok szabályzói érvényesek – ilyen hivatal az Egyesült Államokban az FAA, az Egyesült Királyságban pedig a CAA. A vasúti szabályzók a vasúti közlekedés biztonságosságát biztosítják, a nukleáris szabályzók célja a nukleáris üzemek biztonságosságának igazolása, még a termelés megkezdése előtt. A bankszektorban a szabályzó szerep a nemzeti bankoké: olyan eljárásokat, módszereket dolgoznak ki, amelyek csökkentik a visszaélés lehetőségét, és megvédik a banki ügyfeleket a kockázatos banki gyakorlattól. Ahogyan a szoftverrendszerek az egyes országok kritikus infrastruktúrájában egyre fontosabbak lettek, úgy váltak ezek a szabályzók egyre fontosabbá a szoftverrendszerek biztonságossági és üzembiztonsági esetei szempontjából.

A szabályzók feladata annak ellenőrzése, hogy az elkészült rendszer olyan biztonságos, mint amennyire használható, ezért főként kész projektek esetén alkalmazzák őket. Azonban a szabályzók és a fejlesztők ritkán dolgoznak izoláltan: együtt átbeszélve alakítják ki, hogy a biztonságossági esetnek mit kell tartalmaznia. A szabályzó a fejlesztőkkel együtt vizsgálja a folyamatokat.

Természetesen a szoftver önmagában nem veszélyes, csak akkor, ha olyan nagy, számítógép-alapú szociotechnikai rendszerbe ágyazzuk, ahol a szoftverhibák más eszközök vagy folyamatok sérüléssel vagy halállal járó meghibásodását eredményezik. Épp ezért szoftver biztonságossága mindig egy nagyobb rendszer biztonságosságának részeként értelmezhető. Egy szoftverbiztonságossági eset kialakításakor a szoftverhibákat és a nagyobb rendszer hibáit kapcsolatba kell egymással hozni, és meg kell mutatni, hogy a szoftverhiba vagy nem következik be, vagy ha mégis, akkor a beágyazó rendszerben nem okoz veszélyes rendszerhibát.

Egy biztonságossági eset olyan dokumentumok összessége, amelyek között megtalálható az igazolandó rendszer leírása, a rendszer fejlesztéséhez használt

folyamatok információi, és olyan logikai indoklások, amelyek azt jelzik, hogy a rendszer valószínűleg biztonságos. Bishop és Bloomfield (Bishop és Bloomfield, 1995; 1998) szűkszavúbb megfogalmazása szerint a biztonságossági eset

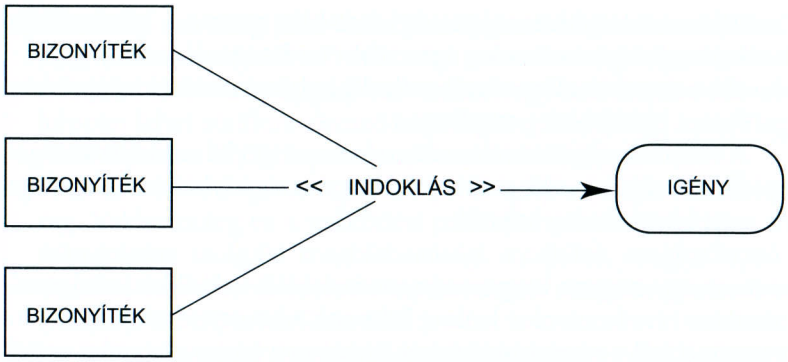
> *egy olyan dokumentált bizonyíték, amely meggyőzően igazolja, hogy egy adott környezet egy adott alkalmazásában a rendszer kellően biztonságos.*

Egy biztonságossági eset felépítése és tartalma függ a rendszer fajtájától és műveleti környezetétől. A 24.11. ábra egy ilyen lehetséges felépítést mutat be.

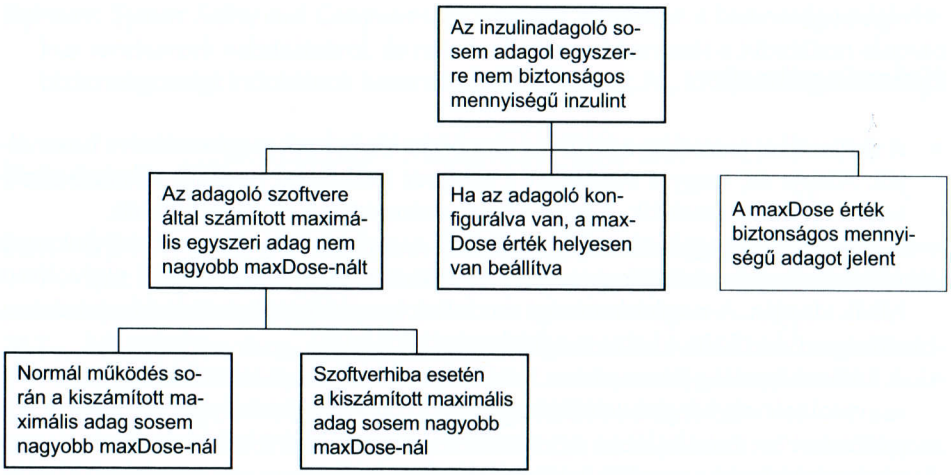
Komponens	Leírás
Rendszerleírás	A rendszer áttekintése, kritikus komponenseinek leírása
Biztonságossági követelmények	A követelményspecifikációból származó biztonságossági követelmények
Veszély- és kockázatelemzés	Az azonosított veszélyhelyzeteket és kockázatokat leíró dokumentumok és a kockázatcsökkentéshez használt mérések
Tervelemzés	Strukturált indoklások összessége, amelyek bemutatják, hogy a terv miért biztonságos
Verifikáció és validáció	Az alkalmazott V & V eljárások leírása, és ahol értelmezhető, a rendszer tesztterve
Áttekintő jelentések	A terv- és biztonságossági átvizsgálásokat rögzíti
Csapat kompetenciái	A biztonságossággal kapcsolatos rendszerek fejlesztésében és validációjában részt vevő csapat tagjainak hozzáértéséről szóló bizonyítékok
Folyamat minőségének biztosítása	A rendszerfejlesztés során végzett minőségbiztosítások feljegyzései
Változtatáskezelési folyamatok	A javasolt változtatásokról, végrehajtott tevékenységekről, és ahol lehetséges, azok biztonságosságának igazolásáról szóló feljegyzések
Kapcsolódó biztonságossági esetek	Az adott biztonságossági esetre kihatással lévő biztonságossági esetek hivatkozásai

24.11. ábra. Egy szoftverbiztonságossági eset komponensei

Egy biztonságossági eset legfőbb összetevője a rendszer biztonságosságáról szóló logikai indoklások halmaza. Ez tartalmazhat abszolút állításokat (az X esemény bekövetkezik/nem következik be) vagy valószínűségi állításokat (az X esemény bekövetkezési valószínűsége 0,Y). A biztonságosságot ezek kombinálásával kell demonstrálni. Ahogyan az a 24.12. ábrán is látható, egy indoklás egy olyan kapcsolat, amely az összegyűjtött bizonyítékok és az elképzelt igények között helyezkedik el. Az indoklás lényegében elmagyarázza, hogy az igény (ami általában valaminek a biztonságossága) a rendelkezésre álló bizonyítékok alapján levezethető. Persze a rendszerek többszintű természete miatt az igények hie-



24.12. ábra. Egy indoklás szerkezete



24.13. ábra. Az inzulinadagoló biztonságosságának követelményhierarchiája

rarchiába szervezhetők. Azt, hogy egy magasabb szintű igény teljesíthető, úgy lehet megmutatni, hogy bemutatjuk az alacsonyabb szintű igények teljesülését. A 24.13. ábrán az inzulinadagoló követelményhierarchiájának egy részlete látható.

Orvosi eszközről lévén szó, az inzulinadagoló rendszerről egy külső tanúsítványt kell készíttetni. Az Egyesült Királyságban árusított orvosi eszközökről például az adott országbeli Medical Devices Directorate-nek (Orvosi Eszközök Főigazgatósága) kell egy biztonsági tanúsítványt kiállítania. A rendszer bitonságosságát igazoló különféle indoklásokat kell megadni. Az inzulinadagoló esetében egy ilyen biztonságossági eset egy része a következő lehet:

Igény: Az adagoló szoftvere által számított maximális egyszeri adag nem nagyobb maxDose-nál.
Bizonyíték: Az inzulinadagoló biztonságossági indoklása (24.7. ábra).
Bizonyíték: Az inzulinadagoló tesztadat-sorozata.
Bizonyíték: Az inzulinadagoló szoftver statikus elemzéséről szóló jelentés.

Indoklás: A bemutatott biztonságossági indoklás szerint a kiszámítható legnagyobb inzulinadag nem több maxDose-nál.
400 teszt során a Dose értéke mindig helyesen került kiszámításra, sosem haladta meg maxDose-t.
A vezérlőszoftver statikus elemzése nem tárt fel rendellenességet.
Összességében elfogadhatjuk, hogy az igényben megfogalmazott követelmény teljesül.

Természetesen ez egy nagyon leegyszerűsített indoklás, valódi biztonságossági eset esetén részletes hivatkozásokat kellene leírnunk a bizonyítékok helyére vonatkozóan. Pontosan ebből a részletezettségből adódóan a biztonságossági esetek nagyon terjedelmes és összetett dokumentumok. Elkészítésüket különféle szoftvereszközök segítik, amelyek elérhetőségeit megtalálhatják a könyv weblapjai között.

Kulcsfogalmak

- A statisztikai tesztelést a szoftver megbízhatóságának megbecslésére használják. Alapja az, hogy a rendszert a szoftver működési profilját tükröző tesztadathalmazzal tesztelik. A tesztadatok automatikusan generálhatók.
- A megbízhatóságnövekedési modellek a megbízhatóságban bekövetkezett változásokat modellezik a tesztelési folyamat során a szoftverből eltávolított hibák alapján. A megbízhatósági modellek használhatók annak előrejelzésére, mikorra érhető el a kívánt megbízhatóság.
- A biztonságosság bizonyítása hatékony technikája a termékbiztonságosság szavatolásának. Megmutatják, hogy egy azonosított veszélyhelyzet sosem következhet be. Ezt általában egyszerűbb elvégezni, mint bebizonyítani, hogy a program teljesíti a specifikációban leírtakat.
- Lényeges, hogy jól definiált, minősített folyamatunk legyen a biztonságoskritikus rendszerek fejlesztéséhez. A folyamatnak tartalmaznia kell a potenciális veszélyhelyzetek azonosítását és monitorozását.
- A védettség validálása elvégezhető tapasztalatalapú elemzéssel, eszközalapú elemzéssel vagy „tigris csoportok" segítségével, amelyek támadásokat szimulálnak a rendszeren.
- A biztonságossági esetek összegyűjtik a rendszer biztonságosságát alátámasztó összes bizonyítékot. Ezekre akkor van szükség, ha egy külső félnek még a használat előtt tanúsítványt kell kiállítania a rendszerről.

További irodalom

„Best practices in code inspection for safety-critical software". Ebben a gyakorlatias cikkben a biztonságosságkritikus rendszerek átvizsgálási irányelveinek egy ellenőrző listája olvasható. (Almeida, J. R. de *és mások*, *IEEE Software*, 20(3), May/June 2003.)

„Statistically scanning Java code: Finding security vulnerabilities". Ez a kiváló cikk általánosságban a biztonsági sebezhetőség elkerülésének rejtelmeibe avatja be az olvasót. Foglalkozik azzal, hogy keletkezik a sebezhetőség, és miképpen lehet statikus elemző segítségével felderíteni azt. (Viega, J. *és mások, IEEE Software*, 17(5), September/October 2000.)

Software Reliability Engineering: More Reliable Software, Faster Development and Testing. Valószínűleg ez a működési profilok használatának és a megbízhatóság felmérésére szolgáló megbízhatósági modellek meghatározó könyve. Statisztikai tesztelésről szerzett tapasztalatok részletes leírásait is tartalmazza. (Musa, J. D., 1998, McGraw-Hill.)

Safety-critical Computer Systems. Ez a kitűnő könyv egy különösen jó fejezetet tartalmaz a formális módszerek biztonságosságkritikus rendszerek fejlesztésében elfoglalt helyéről. (Storey, N., 1996, Addison-Wesley.)

Safeware: System Safety and Computers. Jó fejezetet tartalmaz a biztonságosságkritikus rendszerek validálásáról, és nálam részletesebben szól a hibafákon alapuló biztonságossági indoklások használatáról. (Leveson, N., 1995, Addison-Wesley.)

Feladatok

24.1. Írja le, hogyan validálná azon áruház megbízhatósági specifikációját, amelyet a 9.8. feladat specifikál. Válaszában szerepeljenek a használt validálási eszközök leírásai.

24.2. Magyarázza meg, gyakorlatilag miért lehetetlen validálni a megbízhatósági specifikációkat akkor, ha ezeket a rendszer teljes életciklusából származó nagyon kis számú hiba szempontjából adták meg.

24.3. Az irodalomból háttér-információt nyerve írjon jelentést a vezetőségnek (amelynek nincsen előzetes tapasztalata a területen) a megbízhatóságnövekedési modellek használatáról.

24.4. Etikus-e, ha egy mérnök beleegyezik egy ismert hibákkal rendelkező szoftver átadásába? Jelent-e valami különbséget az, ha az ügyfelet előre figyelmeztették a hibák meglétére? Ésszerű-e a szoftver megbízhatóságára vonatkozó kijelentéseket tenni ilyen körülmények között?

24.5. Magyarázza meg, miért nem jelent garanciát a rendszer megbízhatóságának biztosítása a rendszer biztonságosságára.

24.6. Egy nukleáris hulladékot tároló létesítmény ajtózárás vezérlőmechanizmusát biztonságos működésre tervezték. Biztosítja, hogy a tárolószobába való belépés csak akkor legyen engedélyezve, amikor a sugárpajzsok a helyükön vannak, vagy amikor a sugárzási szint a szobában egy bizonyos érték (dangerLevel) alá esik. Vagyis:

(i) Ha a távolról vezérelt sugárpajzsok egy szobán belül a helyükön vannak, az ajtót egy arra felhatalmazott operátor kinyithatja.

(ii) Ha a sugárszint egy szobában egy meghatározott érték alatt van, az ajtót egy arra felhatalmazott operátor kinyithatja.

(iii) A felhatalmazott operátorokat belépési kód segítségével azonosítja a rendszer.

```
1   entryCode = lock.getEntryCode () ;
2   if (entryCode == lock.authorisedCode)
3   {
4       shieldStatus = Shield.getStatus () ;
5       radiationLevel = RadSensor.get () ;
6       if (radiationLevel < dangerLevel)
7           state = safe ;
8       else
9           state = unsafe ;
10      if (shieldStatus == Shield.inPlace () )
11          state = safe ;
12      if (state == safe)
13          {
14              Door.locked = false ;
15              Door.unlock () ;
16          }
17      else
18      {
19          Door.lock () ;
20          Door.locked = true ;
21      }
22  }
```

24.14. ábra. Ajtózárás-vezérlő

A 24.14. ábrán látható Java-kód szolgál az ajtózárási mechanizmus vezérlésére. Megjegyezzük, hogy a biztonságos állapotnak annak kell lennie, amikor a belépés nem engedélyezett. Készítsen biztonságossági indoklást, amely megmutatja, hogy ez a kód potenciálisan nem biztonságos. Módosítsa a kódot úgy, hogy biztonságos legyen.

24.7. Az adag kiszámításának a 10. fejezetben szereplő specifikációját felhasználva (10.11. ábra) írja meg a computeInsulin Java-metódust, amelyet a 24.6. ábrán is használunk. Alkosson informális biztonságossági indoklást arról, hogy ez a kód biztonságos.

24.8. Mondja el, hogyan végezné egy jelszóvédelmi rendszer validálását egy ön által fejlesztett alkalmazás számára. Fejtse ki minden olyan eszköz szerepét, amelyet ön hasznosnak ítél.

24.9. Miért szükséges a rendszer változásainak részleteit a szoftver biztonságossági esetei között feltüntetni?

24.10. Milyenfajta rendszerek esetén van szükség biztonságossági esetekre? Soroljon fel négyet.

24.11. Tegyük fel, hogy tagja volt egy vegyi üzem számára szoftvert fejlesztő csapatnak. A szoftver rosszul működött, és súlyos szennyezést idézett elő. Főnöknőjével interjút készítenek a televízióban, melyben ő kijelenti, hogy a validálási folyamat minden részletre kiterjed, és a szoftverben nincs hiba. Azt állítja, hogy a probléma az üzem működési eljárásmódjainak a következménye. önt is megkereste egy napilap a véleményéért. Fejtse ki, hogyan ajánlott egy ilyen interjút kezelnie.

24.12. Milyen érvek vannak a szoftvermérnökök engedélykötelessége mellett, illetve ellen?