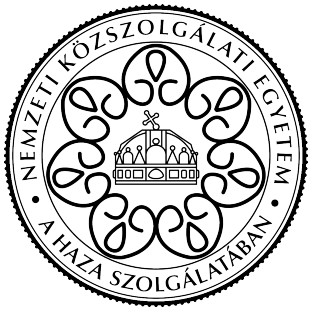
Nemzeti Közszolgálati Egyetem
Katonai Műszaki Doktori Iskola



Mészáros Gergely

Létfontosságú Információs Rendszerelemekben alkalmazott nyílt forráskód és szabad szoftver rendszerszemléletű biztonsági analízise

Doktori (PhD) értekezés tervezet

Témavezetők:

Prof. Dr. Haig Zsolt mk. ezredes **Dr. Muha Lajos mk. alezredes**

....

Budapest, 2020

Tartalomjegyzék

Bevezetés

0.1 Tudományos probléma megfogalmazása, időszerűsége

Korunk sokszorosan összetett, egymásba fonódó technológiákon alapuló információs társadalma egyre komolyabb kihívás elé állítja a biztonsági szakembereket. A rendszerek összetettsége, az alkalmazott eszközök és módszerek száma folyamatosan nő, lassan olyan mértéket érve el, amelyet az emberi elme már képtelen befogadni. Elég csak a képzésre gondolni. Míg alig néhány évtizede egyszerűen informatikusokat (sőt elektromérnököket) képeztek, ma már tengernyi szakirány létezik, külön szakmává válik a felhőtechnológia, adatelemzés, mesterséges intelligencia kutatás vagy a kiberbiztonság, de lassan azt is tovább bonthatnánk kriptográfiára, hálózatbiztonságra, üzemeltetésre, kockázatmenedzsmentre és sok egyéb területre. Az informatika exponenciális tudásnövekedése folytán ma már elképzelhetetlen, hogy valaki informatikai polihisztor legyen.

Az összetett világ összetett fenyegetéseket jelent, ha pedig mindezt összevetjük a mindenhová begyűrűző információs technológiával, meglehetősen veszélyes elegyet kapunk. Különösen veszélyeset, ha olyan területről van szó, ahol emberéletek foroghatnak kockán, ahol egy esetleges hiba vagy támadás jelentős kárt képes okozni, azaz a Kritikus Infrastruktúrák más néven Létfontosságú Rendszerelemek területén.

A különféle Létfontosságú Rendszerelemek egymással összetett függőségi viszonyban állhatnak, így az sem feltétlen szükséges, hogy maga a rendszer kritikus legyen, elegendő, ha kiesésével más kritikus rendszer működése válik lehetetlenné, így a működési zavar térben és időben szétterjedve a lakosság ellátásában, a gazdaság vagy kormányzat működésében komoly problémákat okozhat. [1]

Az informatika fejlődésének egyik szembetűnő jelensége a nyílt forrású technológiák egyre erősödő ter-

jedése. Néhány évtizede műkedvelők játékának esetleg naiv ideológiának tartott fejlesztési modell üzleti támogatók seregét maga mellé állítva az ipar egyik legfontosabb tényezőjévé nőtte ki magát.

A Nyílt Forrás fogalom az utóbbi években jelentős utat járt be a kezdeti, első sorban ideológiai indíttatású Szabad Szoftver elképzeléstől számítva. Ez a piaci szereplők által támogatott, modern változat már egyértelműen mainstream technológiának számít, üzletileg elfogadható sőt kívánatos elemnek, amely gyakran alkalmazott megoldás mind a szoftverfejlesztés mind a terjesztés során. Használatának előnyei nyilvánvalóak. Megengedi a hozzáférést a forráskódhoz, lehetővé teszi a származtatott termékek terjesztését és jelentős befektetés nélkül is elérhetővé teszi az élvonalbeli technológiákat. Ezekkel a potenciális előnyökkel nehéz versenyezni.

Szinte minden területen nyílt forrású fejlesztésekkel találjuk szembe magunkat, nyílt fejlesztést vezeti ma már előremutatónak, "trendinek" számít, olyan vezető tech cégek igyekeznek meggyőzni bennünket a nyílt forrással való szoros barátságukról mint a Google, Facebook vagy akár a Microsoft, amely egy évtizede még teljesen más hangnemet ütött meg a nyílt modellel kapcsolatban. A folyamatot olyan sikertörténetek kísérik mint a szintén FLOSS[1] terméknek számító Linux kernelen alapuló Android, melynek piaci részesedése az előrejelzéseknek megfelelően [2] mára világszinten elérte a 77%-ot, míg Ázsiában meghaladja a 84%-ot [3, 4]. Nagyon erős nyílt forrás jelenléte az IoT[2] technológiák világában [5], a webszerverek és általában a szerver üzemeltetés területén. A valamikor egyetlen – bár igen népszerű – webszerver köré szerveződő Apache™ alapítvány ma már közel kétszáz projektet gondoz számos területen. Sok közülük egyáltalán nem nevezhető jelentéktelennek, az Apache™ Hadoop® piaca például 2022-re az előrejelzések szerint meghaladja a 50.0 milliárd dollárt[6]. A böngészők területén a nyílt forrás töretlenül erősödik [7] az egyetlen jelentős zárt forrású szereplő, az Internet Explorer az Edge böngészővel együtt is alig éri el a 16%-os piaci részesedést [8], ami jelentős változás a néhány évvel korábbi állapothoz képest. Hosszasan lehetne sorolni a példákat.

A hozzáállás megváltozásának oka abban keresendő, hogy maga a nyílt modell is megváltozott. Az üzleti világgal való együttélése során olyan szimbiózis alakult ki amely egyesítette a két világ előnyeit lehetővé tette a nagymértékű fejlődést és elterjedést. Ezt az "új" nyílt forrás fogalmat gyakran nyílt forrás 2.0,

[1] Free Libre and Open Source software. Az értekezésben használt értelmezése a 1.2. fejezetben olvasható.
[2] Internet of Things, a "dolgok internete", egymással kommunikálni képes fizikai eszközök (épületek, járművek, háztartási és megfigyelő eszközök stb.) komplex hálózata.

támogatott nyílt forrás vagy hibrid nyílt fejlesztői közösség[3] néven találjuk meg a szakirodalomban [9].

A támogatott nyílt forrás elterjedése végül maga után húzta a klasszikus modellt is. A nyílt forrás mint használható alternatíva elfogadottsága nagy mértékben megnőtt és implicit vagy explicit módon begyűrűzik a korábban kizárólag üzleti termékek által uralt területekre is. A programfejlesztésben ugyanis napjainkban bevett szokás a nyílt forrásból származó komponensek extenzív használata – ezáltal a nyílt forrás – közvetett módon a technológia minden területére kihat.

Joggal merül fel tehát a kérdés, hogy vajon ez a megváltozott felállás milyen hatást gyakorol az információs rendszerek biztonságára, van-e jelentős eltérés az üzleti rendszerekhez és komponensekhez képest valamint szükséges-e változtatni az alkalmazott védelmi eljárásokon.

Természetesen az egész problémakört nagyon egyszerűen szőnyeg alá lehet söpörni annyival, hogy a szervezet nem használ semmilyen nyílt forrású terméket, tehát az üggyel nem kell foglalkozni. Ez a megközelítés azonban véleményem szerint ma már elégtelen, sőt veszélyes. A szervezet nem biztos, hogy tudatában van annak, ha FLOSS elemeket használ. Az informatikai fejlesztésekben rendkívül széles körben elterjedt FLOSS komponens használat következtében a szervezet beszállítói és fejlesztői nagy valószínűséggel használnak FLOSS komponenseket, továbbá az sem zárható ki, hogy a szervezet alkalmazottai rendszeren kívül használjanak ilyen terméket. Emiatt akkor is kell foglalkozni a FLOSS kérdésével ha a szervezet teljesen elhatárolódik a FLOSS felhasználástól, legalább annyiban, hogy ezt az elhatárolódást szabályozás révén a gyakorlatban is biztosítani lehessen.

Összefoglalva, a FLOSS biztonsági hatásait célzó rendszer szintű kutatás elvégzését három tényező indokolja:

- Az egyre erősödő kiberfenyegetés;
- FLOSS felhasználás növekvő mértéke és implicit jellege;
- meglévő FLOSS specifikus szabályozás hiánya vagy elégtelen volta.

A FLOSS fejlesztési módszertan vonzó lehetőség mind a technológia óriások, mind a frissen induló startupok számára [10, 11]. Függetlensége, nyíltsága és átláthatósága révén használatának igénye egyre gyakrabban felmerül a közigazgatásban is [12].

[3] Az angol szakirodalomban Open Source 2.0 avagy Hybrid Open-Source (HOSS)

A szoftver vagy komponens teljes átláthatósága komoly előnyöket is hordoz, sőt, a kormányzati szférában idővel akár követelmény is lehet. A könnyű elérhetőség és választék nagy mértékben lerövidíti a fejlesztési időt, amit egy adott piaci helyzetben egyszerűen nem lehet figyelmen kívül hagyni. Ilyenformán a FLOSS felhasználás stratégiai cél is lehet. Tapasztalataim szerint viszont a szervezetek legtöbbször nem rendelkeznek célzottan FLOSS specifikus szabályozással. A FLOSS fejlesztési környezete vagy felhasználásának körülményei ugyanakkor olyan mértékben eltérhet a megszokottól, hogy a meglévő szabályozás már nem alkalmas annak kezelésére. Ez szervezeti, állami vagy nemzetközi szinten egyaránt igaz lehet.

Véleményem szerint a biztonság kérdését nem lehet kizárólag műszaki oldalról megfogni, annak ellenére, hogy az érzékelhető védelmi intézkedések során ezek a leglátványosabb elemek. A kezdeti egyszerű fizikai biztonság fogalom ma már túlhaladott, a klasszikus biztonságpolitikai felfogás szerint a biztonságnak öt dimenzióját különíthetjük el:

- politikai biztonság;
- katonai biztonság;
- társadalmi biztonság;
- gazdasági biztonság és
- környezeti biztonság.

Ezek a dimenziók szoros kapcsolatban állnak egymással és kölcsönhatások léphetnek fel közöttük, ennélfogva az egyes területeken jelentkező veszélyek más dimenziókra is átterjedhetnek amennyiben a reakció nem megfelelő [13]. Ezeknek a dimenzióknak a hatása akkor is érzékelhető, ha a biztonság fogalmát a kutatás tárgyát képező informatikai biztonságra szűkítjük le.

A 2013. évi L. törvény megfogalmazása szerint a kiberbiztonság “a kibertérben létező kockázatok kezelésére alkalmazható politikai, jogi, gazdasági, oktatási és tudatosságnövelő, valamint technikai eszközök folyamatos és tervszerű alkalmazása, amelyek a kibertérben létező kockázatok elfogadható szintjét biztosítva a kiberteret megbízható környezetté alakítják a társadalmi és gazdasági folyamatok zavartalan működéséhez és működtetéséhez” [14].

A biztonság tehát összetett fogalom, emiatt a FLOSS technológiák hatásának vizsgálatát is komplex megközelítéssel kell elvégezni és nem elegendő pusztán a technikai megoldásokra fokuszálni.

Az nyílt modell információs rendszerek biztonságára gyakorolt hatását nem lehet anélkül elemezni, hogy pontosan ismernénk azokat a hatáspontokat ahol és amilyen módon az információs rendszer kapcsolatba kerülhet FLOSS rendszerekkel. A kezdeti szabad szoftver fogalmától napjainkra egy összetett nyílt fejlesztési modellig jutottunk el, amely technológiánkat nyíltan vagy rejtetten számos helyen átszövi. Ennélfogva a célkitűzéseim között szerepelt, hogy a vizsgálatba ne pusztán az "ingyenesen felhasználható" Szabad Szoftverek köre kerüljön be, hanem minden olyan tény és információelem, amely a nyílt fejlesztési modell sajátosságaiból adódóan publikusan elérhető. A Szabad Szoftverek mellett ide értve FLOSS komponenseket, forrástárakat, szoftvertárolókat a nyílt fejlesztés során létrehozott és felhaszált minden metaadatot sőt, a vizsgálat tárgya kell legyen a fejlesztői közösség kommunikációja, szociális és gazdasági viszonyai és az egyes szereplők egymásra gyakorolt hatása is.

Tekintettel arra, hogy a FLOSS felhasználás több módon is megvalósulhat, valamennyi esetet érdemes vizsgálat alá vonni. A szervezet lehet közvetlen vagy – beszállítóin keresztül – közvetett felhasználó, lehet közösségi partner és a fejlesztési folyamat részeként saját vagy piaci célokra terméket előállító szereplő is. Fontos jellegzetesség a projektben való közvetlen résztvétel vagy a saját belső fejlesztésben történő felhasználás. Egyetértek Krasznay-val abban, hogy a fejlesztői környezet biztonsága olyan terület amelynek nincs kiforrott hagyománya hazánkban [15] ezért ezzel a kérdéssel kiemelten érdemes foglalkozni.

Jelenleg nincs olyan általam ismert kutatás amely a nyílt fejlesztési modell módszertanának biztonsági hatásait komplex módon elemzi. Sok tanulmány foglalkozik a nyílt forrás minőségbiztosításának kérdésével, még több a vélt vagy valós előnyökkel és hátrányokkal, de nyílt forrású fejlesztési módszertan biztonsági hatásainak megértéséhez nem elegendő egyetlen oldalról megközelíteni a kérdést.

A FLOSS biztonságra gyakorolt hatása jelentős, sokrétű és rejtett lehet. Mind tudományos mind gyakorlati szempontból érdekes, hogy a jelenség milyen mértékű és horderejű hatást gyakorol a magas biztonsági követelményeket támasztó rendszerek, különösen a Létfontosságú Információs Infrastruktúrák biztonsági szintjére. Vajon elegendő-e a meglévő szabályozás, képesek-e a szervezetek megfelelni a változó körülményeknek és helyén tudják-e kezelni a FLOSS felhasználásból eredő esetleg szokatlan kockázatokat?

A kutatás tervezése során ezekre a kérdésekre igyekeztem választ találni.

0.2 Célkitűzések

A kutatás alapvető célkitűzése annak meghatározása volt, hogy a magas biztonsági követelményeket támasztó információs rendszerek – különösen és elsősorban a Létfontosságú Információs Rendszerelemek (továbbiakban LIRE) – biztonságára milyen hatást gyakorolhat a FLOSS technológiák napjainkban tapasztalható előretörése.

A kutatás alapkérdésének megválaszolásához szükség volt a kérdés felbontására.

A cél eléréséhez meg kell határozni, hogy milyen utakon kerülhet kapcsolatba a FLOSS és a Létfontosságú Rendszerelem információs rendszere, továbbá milyen szintű együttműködés szükséges ahhoz, hogy a biztonságra gyakorolt hatás már érezhető legyen. Továbbá, be kell azonosítania azokat a FLOSS sajátosságokat, amelyek konkrét biztonsági hatást képesek gyakorolni az információs rendszerre. A sajátosságok és a hatáspontok meghatározása után a következő lépés a konkrét biztonsági hatások és azok kiaknázását vagy elkerülését célzó műveletek feltérképezése és rendszerezése. Végül, véleményem szerint úgy lehet megítélni a FLOSS jelenség által kifejtett hatások jelentőségét, ha sikerül felderíteni a meglévő szabályozás által már lefedett és le nem fedett problémákat, azaz be tudom azonosítani azokat a pontokat ahol a FLOSS ténylegesen ki is tud fejteni pozitív vagy negatív hatásokat. Az utolsó célom tehát a beazonosított biztonsági hatásokat a meglévő védelmi intézkedésekkel és szabályozással összevetve következtetéseket levonni a nyílt fejlesztési modell ténylegesen realizálódó biztonsági hatásait illetően.

A fentieken túlmenően a kutatás reprodukálhatósága érdekében célom volt jól definiált és dokumentált módszertant alkalmazni és az Open Science irányelvei mentén valamennyi kutatási anyagot és adatot publikusan elérhetővé tenni. Ezáltal remélhetőleg a kutatás minden lépése nyomon követhető és az összegyűjtött információ mások számára is könnyen felhasználhatóvá válik.

A fentiekkel összhangban a következő kutatási célkitűzéseket fogalmaztam meg:

- KC-1: Meghatározni, hogy a Létfontosságú Információs Rendszerelemek milyen módon kerülhetnek kapcsolatba a FLOSS elemekkel.
- KC-2: Meghatározni és rendszerezni a nyílt modell azon sajátosságait, amelyek a befolyásolhatják az informatikai biztonságot.

  - KC-2.1: Felmérni, hogy a FLOSS egyes hatásai mennyira kutatottak a különféle területeken.
  - KC-2.2: Létrehozni a FLOSS jellegzetességeit összefoglaló, minél átfógóbb rendszert.
  - KC-2.3: Beazonosítani azokat a sajátosságokat, amelyek hatással vannak a biztonságra.
- KC-3: Meghatározni azokat a pozitív és negatív hatásokat amelyek a Létfontosságú Rendszerelemek biztonságát befolyásolhatják.
- KC-4: Intézkedések formájában javaslatot tenni a negatív hatások elkerülésére és a pozitív hatások kiaknázására.
- KC-5: Az eredményeket az érvényes hazai szabályozással összevetve következtetéseket levonni a FLOSS LIRE felhasználhatóságára vonatkozóan.

0.3 Kutatási hipotézisek

A kutatás céljainak megfelelően a 0.4 fejezetben részletesen tárgyalt kutatási kérdések alapján a következő kutatási hipotéziseket állítottam fel:

H1. Feltételezem, hogy a nyílt fejlesztési modell és az így előállított termék olyan egyedi tulajdonságokkal rendelkezik, amely sajátos módon befolyásolhatja a Létfontosságú Rendszerelemek biztonságát.

H2. Feltételezem, hogy definiálhatóak olyan ellenintézkedések, amelyek a FLOSS felhasználás sajátosságaiból eredő biztonsági problémák okozta kockázatot mérsékelni tudják.

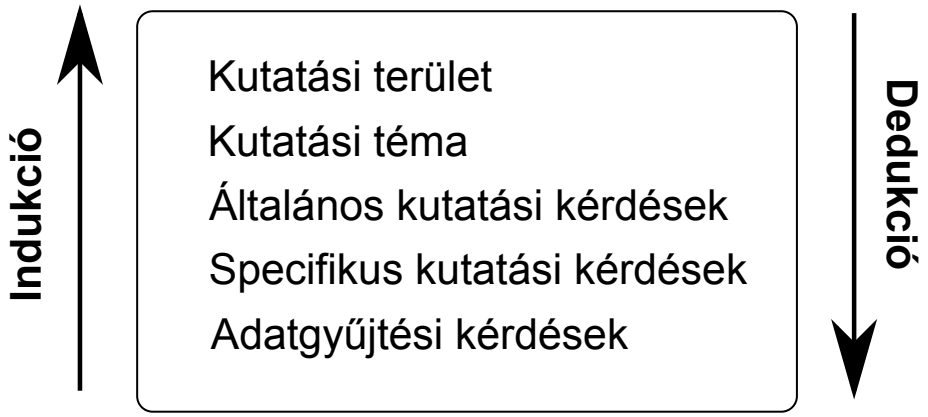
H3. Feltételezem, hogy az eltérő minőségű és forrású FLOSS termékek bizonyos csoportját megfelelő ellenintézkedések foganatosítása mellett a legmagasabb biztonsági elvárásokat támasztó Létfontosságú Rendszerelemek területén is fel lehet használni.

0.4 Kutatási módszer

A kutatás célkitűzéseivel összhangban olyan módszert igyekeztem alkalmazni, amely átfogó képet nyújt teljes kérdéskörről, ugyanakkor módszeres elemzést tesz lehetővé, hogy a szubjektivitás mértékét a lehetőségekhez képest alacsony szinten tartsam. A választott módszertan tehát legyen:

- rendszerszemléletű,
- reprodukálható,
- és szisztematikus.

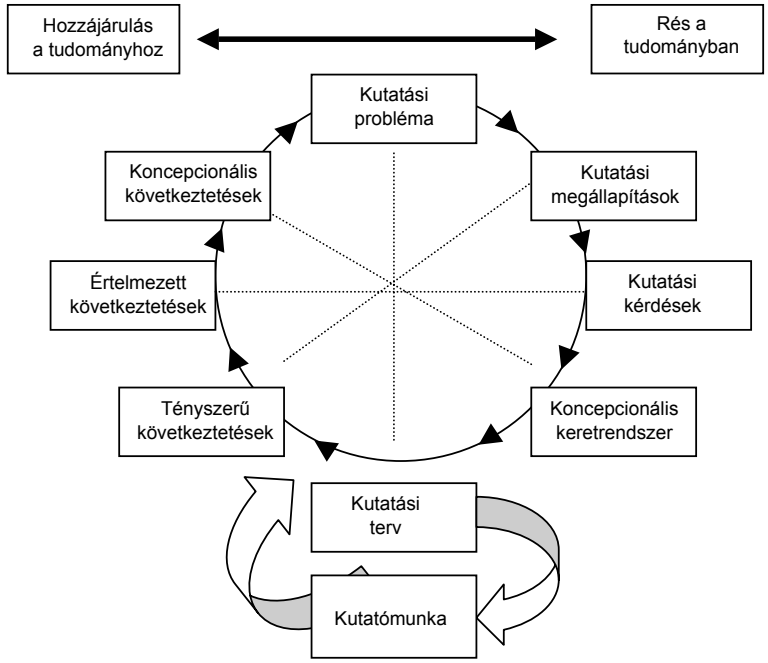
Punch szerint az empirikus elemzések adatgyűjtéséhez hierarchikus kérdésmegfogalmazás alapján juthatunk el [16]. A szintek között dedukció és indukció módszerével haladva lehet a generalizált kérdésből kutatási kérdéseket alkotni majd az összegyűjtött eredményekből következtetéseket levonni (Punch hierarchiáját a 0.1. ábra mutatja be). Punch elgondolása szerint a kutatás tervezésének leghatékonyabb módja a "mit?" kérdésének felvetése a "hogyan?" előtt, azaz azt javasolja, hogy a kutató tisztázza a kutatási stratégiát mielőtt eldöntené, hogy mely módszerek alkalmasak a stratégiai célok elérésére.



0.1. ábra: Punch fogalom-hierarchiája (Saját szerkesztés, Punch nyomán [16])

Leshem és Trafford szerint a doktori kutatási folyamat ciklikus rendszerként képzelhető el, ahol a tudomány terültén feltárt hiányosságokat és az arra adott válaszokat célzó lépések sorozata váltja egymást (lásd 0.2. ábra). A kutatási problémától a kutatási tervig a tudomány hiányosságait célzó egyre konkrétabb lépéseken keresztül jutunk el, míg az utolsó három tevékenység három különböző szintű eredménnyel járul hozzá a tudományhoz. Az egyes lépések között belső összefüggések (szaggatott vonalak) biztosítják a kutatás önkiegyensúlyozó jellegét[17].

A javasolt módszertan mentén haladva először a megfogalmazott kutatási kérdéseket mutatom be, majd a koncepcionális keretrendszert végül az alkalmazott módszereket és a kutatási szerkezeti felépítését



0.2. ábra: Doktori kutatás módszertanának vázlata (Leshem és Trafford ábrája nyomán [17])

0.4.1 Kutatási kérdések

Az alapkérdés megválaszolásának érdekében az alapkérdést felbontandó a következő kutatási kérdéseket fogalmaztam meg:

- **RQ1:** Vannak-e és melyek a nyílt fejlesztési modellből származó termékeknek olyan egyedi sajátosságai amelyek hatással lehetnek a biztonságra?
- **RQ2:** A magas biztonsági követelményű rendszerek hogyan kerülhetnek kapcsolatba FLOSS elemekkel?
- **RQ3:** A FLOSS sajátosságok milyen hatást gyakorolnak a Létfontosságú Információs Rendszerelemekre?
- **RQ4:** A negatív hatásokat milyen szabályozásbeli, szervezeti és technikai óvintézkedések foganatosításával lehet elkerülni, a pozitív hatásokat hogyan lehet kihasználni?

Az első kérdés esetében kézenfekvő, hogy ilyen nagy területet átfogó analízist csak szekunder kutatás segítségével, minél nagyobb számú kutatási anyag tartalmi elemzésével lehet elvégezni. Ahhoz, hogy az RQ1-re kapott válasz érvényességét meg lehessen becsülni először tehát meg kell állapítani, hogy a tudományos kutató közösség milyen mélységben foglalkozott a kérdéskörrel. Ezt követően fel kell mérni,

hogy a FLOSS egyáltalán milyen (nem feltétlenül biztonság specifikus) sajátosságokkal rendelkezik, mert csak így biztosítható a célkitűzésben megfogalmazott teljeskörűség, objektívitás és függetlenség. Végül az összes azonosított sajátosságból ki lehet zárni azokat amelyeknek nincs biztonsági vonatkozása és rendszerbe lehet foglalni a biztonsági hatást gyakorló FLOSS sajátosságokat.

Ennek az elképzelésnek megfelelően az RQ1-et az alábbi kérdésekre bontottam:

- **RQ1.1:** A FLOSS hatásai mennyire kutatottak a információbiztonság egyes területein?
- **RQ1.2:** Melyek a nyílt fejlesztési modell egyedi jellegzetességei?
- **RQ1.3:** Mely sajátosságok esetén azonosítható konkrét biztonsági hatás?

A precíz válasz érdekében az RQ4 szintén további bontást igényelt. Itt a következő kérdéseket fogalmaztam meg:

- **RQ4.1:** Milyen intézkedésekkel csökkenthető a FLOSS használatának kockázata?
- **RQ4.2:** Melyek a kutatóközösség által javasolt módszerek?
- **RQ4.3:** Mely biztonsági problémák kezelésére nincs bevett gyakorlat?

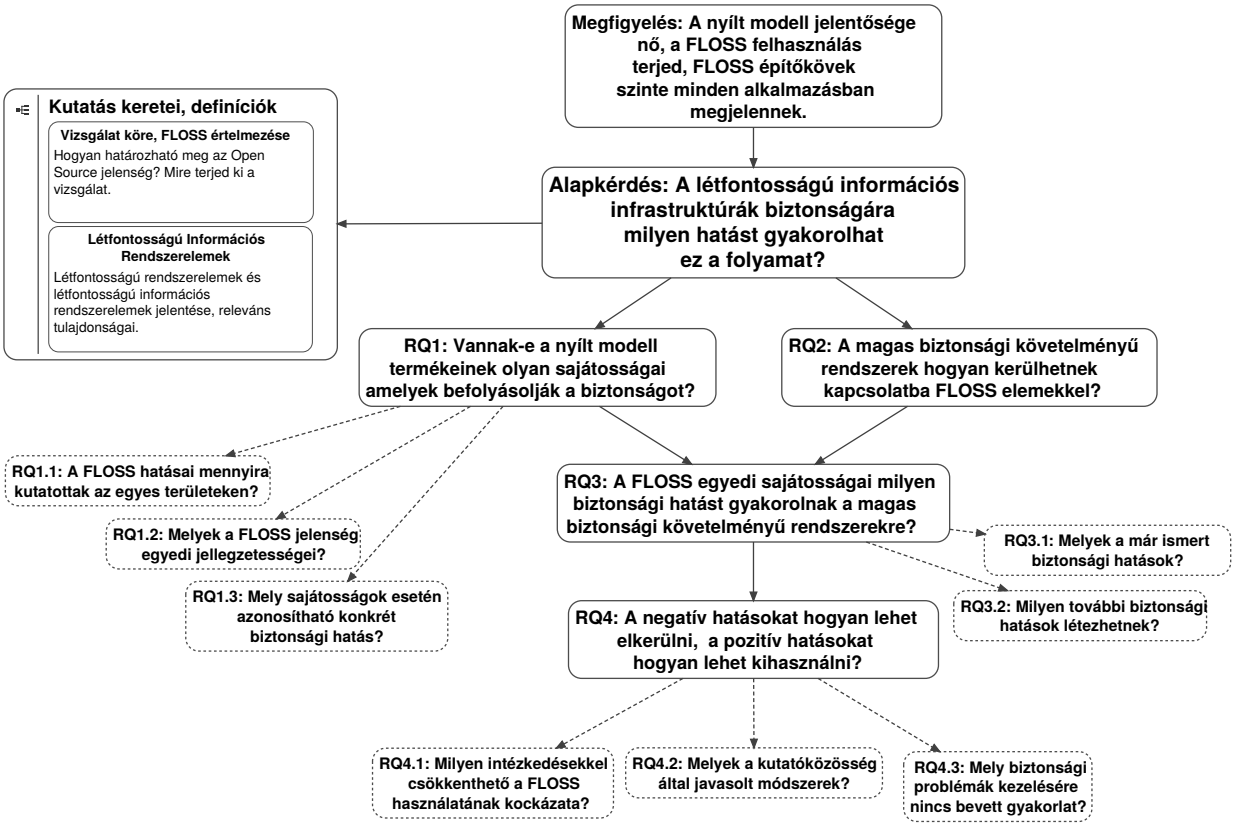
A kutatási kérdések teljes hierarchiáját a 0.3. ábra mutatja be.

A 0.2 fejezetben bemutatott célkitűzéseket (KC-[1-4]) Punch kutatási módszertanát követve a fent bemutatott kérdések alapján fogalmaztam meg. Az értekezés további részében a célkitűzések jelölésrendszerét használom.

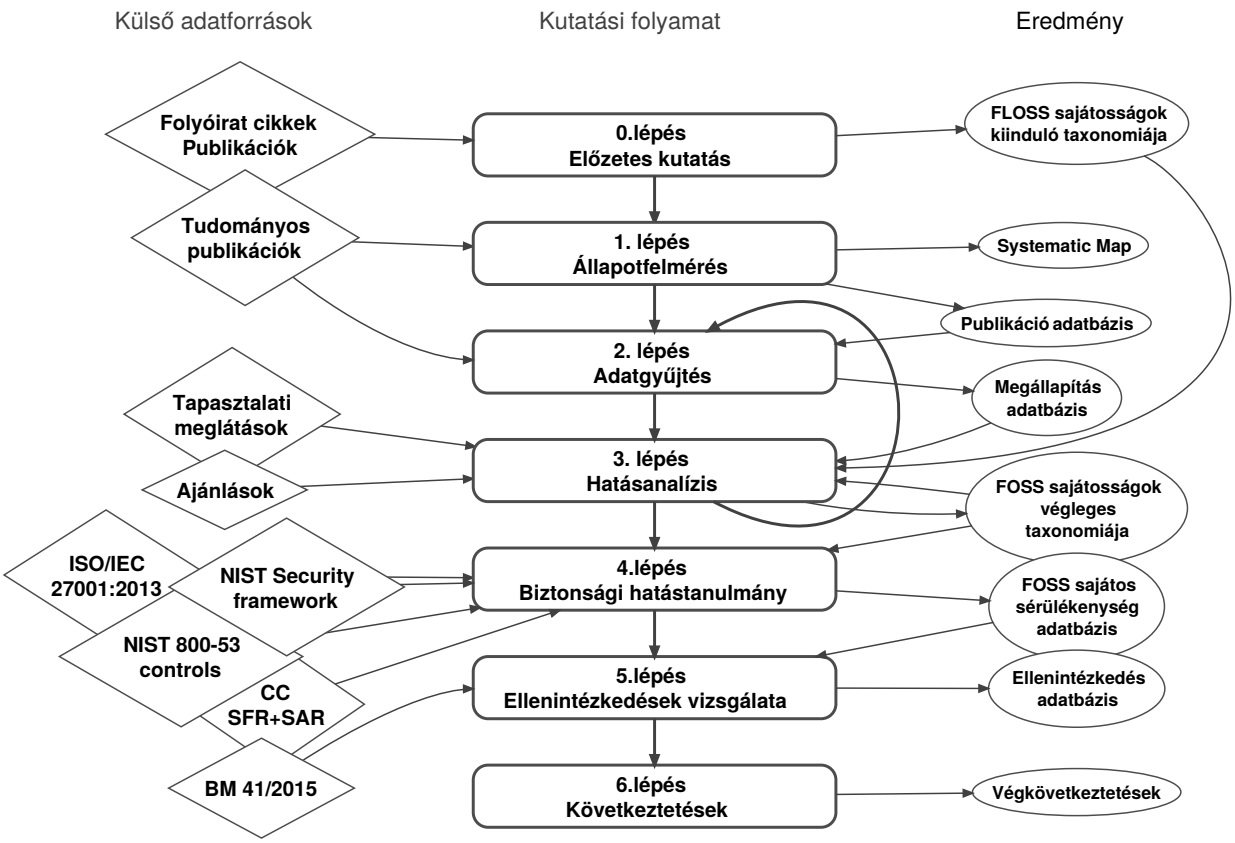
0.4.2 Koncepcionális keretrendszer

Az alkalmazott kutatási módszertan következő lépéseként a 0.4. ábrán bemutatott koncepcionális keretrendszert állítottam össze. Innen kiolvashatóak a kutatás szükséges lépései, a kutatás során feldolgozott információforrások és a várt eredmények.

Az ábra első oszlopában találhatóak a felhasznált információforrások. A kutatás elsődleges információforrása a feldolgozott cikkekből és tanulmányokból manuálisan kigyűjtött adatok, de a hatásanalízis során meglévő ajánlásokat, illetve létező, aktívan használt keretrendszerek (ISO/IEC 27001, NIST 800-53, Common Criteria) javaslatait is figyelembe vettem. Ezáltal biztosított a kutatási célokban megfogalmazott



0.3. ábra: Kutatási kérdések hierarchiája (szerkesztette a szerző)



0.4. ábra: A kutatás koncepcionális keretrendszere (szerkesztette a szerző)

(KC-2) teljeskörűség mind a sajátosságok, mind a biztonsági hatások tekintetében.

A második oszlopban lekerekített téglalpokkal jelölt kutatási folyamat lépéseiből látható, hogy a kutatás adatgyűjtő-elemző része ciklikus jellegű, azaz a hatásanalízis során finomított taxonomia és a biztonsági hatások bővülő kategóriái az adatgyűjtés során újra felhasználásra kerültek. A ciklikus megvalósításra részben azért volt szükség, mert a cimkézés során létrehozott új kategóriákat a már feldolgozott publikációk anyaga esetében is értelmezni kellett, másrészt az analízis-szintézis szakasz elég hosszú időt vett igénybe, és az időközben megjelent új publikációkat folyamatosan rendszerbe kellett illeszteni.

A harmadik oszlopban feltüntetett eredmények olyan dokumentumokat és adatbázisokat jelölnek, amelyek a kutatás további fázisaiban, más kutatásokban vagy a gyakorlatban felhasználható információt és adatokat tartalmaznak.

0.4.3 Alkalmazott kutatási módszerek

Az KC-2.1 elérésre Petersen által javasolt szisztematikus feltérképezés[4] módszerét alkalmaztam [18, 19].

A szisztematikus térképezés módszere széles körben alkalmazott praktikus eszköz a szoftvermérnöki területek osztályozási feladataira és a kutatás struktúrájának felmérésére. Ez az analízis a publikációk kategóriánkénti sűrűségére koncentrál, így meghatározható a terület kategóriánkénti becsült fedettsége. A klasszikus szisztematikus térképezés nem merül el a részletekben, a publikációkat nem elemzi részletesen. A javasolt módszerhez képest esetemben mélyebb elemzésre volt szükség – közeledve a klasszikus szisztematikus forráselemzéshez [20] – ugyanis a publikációk alkalmazott módszertanát és eredményeinek típusát is meg akartam határozni.

A FLOSS sajátosságok kategóriáihoz az előzetes kutatások során [21] előállított FLOSS sajátosság taxonomia vázlatot használtam fel amelyet a az összegyűjtött anyagok segítségével fokozatosan pontosítottam.

A szisztematikus feltérképezés alapját képező gyűjtőmunka kettős célt szolgált. Egyrészt a szisztematikus feltérképzés segítségével meghatározhatóvá vált, hogy mely területek milyen mértékben kutatottak, azaz az eredmények várható megbízhatósága és teljessége milyen szintű lesz az egyes területeken, másrészt az összegyűjtött és felcímkézett forrásanyag alapját képezhette a kutatás következő analitikus fázisának.

[4]Systematic Mapping Study

Minthogy a feltérképezést 2016-ban végeztem és a teljes anyagmennyiség feldolgozása illetve az analízis sok időt vett igénybe, az eredeti publikáció adatbázist azonos keresési metodika használata mellett ismétlődő frissítések során bővítettem. Ennek megfelelően míg az előzetes SMS kutatás csak 2016-ig tartalmaz anyagokat, az analízisben már a legfrissebb publikációk is szerepelnek.

Az KC-2.2, KC-2.3 és KC-3 célok elérése érdekében az analízis-szintézis módszertanát alkalmaztam. Az első fázisban összegyűjtött és folyamatosan kiegészített dokumentumokból felépítettem a FLOSS sajátosságainak lehető legteljesebb modelljét, majd a modell alapján meghatároztam a biztonsági hatásokkal kapcsolatos jellemzőket illetve – amennyiben voltak ilyenek – a javasolt megoldásokat. Az KC-2 analízis kimenetének és a KC-1 eredményeinek szintézisével határoztam meg a KC-3 alatt megfogalmazott területeket és hatáspontokat, illetve becslést adok a be nem azonosított sajátosságok és védelmi intézkedések számát illetően.

Végül a FLOSS sajátosságok és a hatáspontok modelljét, valamint a magas biztonsági rendszerekre vonatkozó előírásokat összevetve szintézis segítségével határoztam meg a KC-4 alatt megfogalmazott lehetséges ellenintézkedéseket. A védelmi intézkedések kialakítását a NIST 800-53 security overlay mechanizmusa inspirálta. A NIST security overlay egységes sablon kialakítását javasolja a speciális követelményeket támasztó ágazatokhoz és szervezetek számára [22]. Hasonló elképzelés mentén terveztem a nyílt forrást alkalmazó és magas biztonsági követelményeknek megfelelni kívánó szervezetek számára olyan intézkedés overlay-t képezni, amely a meglévő szabályozást kiegészítve felhívja a figyelmet a FLOSS esetében eltérően vagy különös figyelemmel kezelendő pontokra.

A NIST 800-53 overlay elképzelésével ellentétben helyhiány miatt csak a különbözetet adom meg, amelyekkel az eredeti szabályokat szükség esetén ki lehet egészíteni, értelemszerűen az általános esetben érvényes eljárásokat továbbra minden esetben alkalmazni kell.

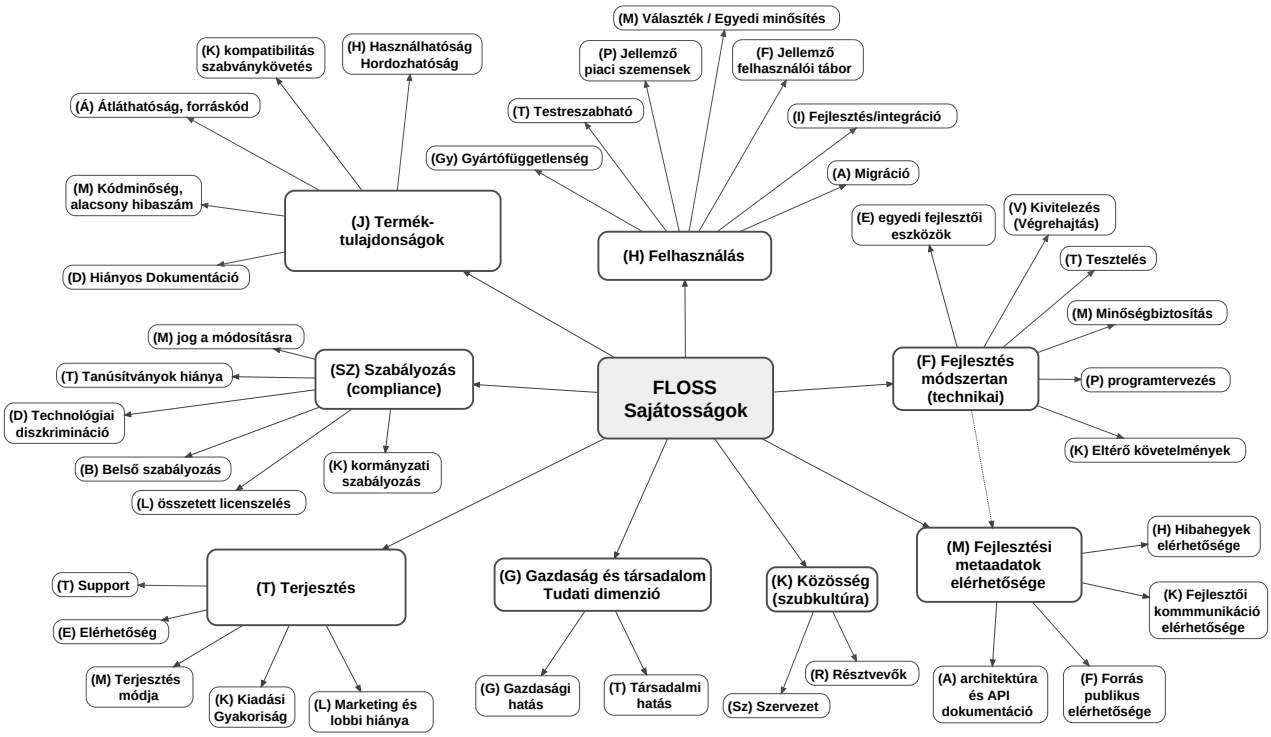
0.4.4 Az értekezés felépítése, jelölésrendszere

Az első fejezet a kutatás tárgyával és irodalmával foglalkozik. Itt definiálom a kutatás témájának kereteit és a vizsgált területeket. Ugyanitt találhatóak az irodalomkutatás eredményei amelyek a kutatási célkitűzésekkel összhangban az egyes biztonsági területek vizsgálatának alaposságát kívánják meghatározni, amely alapján az eredmények teljeskörűsége végül becsülhető. Az első fejezet végén a Létfontosságú Rendszer-

elemek és a nyílt forrás kapcsolódási pontjait vizsgálom, amely alapul szolgál a következő fejezetekben tárgyalt biztonsági analízishez.

A második fejezet a nyílt forrás egyedi jellegzetességeivel foglalkozik, az RQ1-ben megfogalmazott kérdésre adva választ. Az itt azonosított egyedi sajátosságok alapján válik végül lehetségessé a RQ3 kérdésének megválaszolása.

A második fejezetben alkalmazott rendszertant a 0.5. ábra mutatja be. Az egyes fejezeteket a könnyű azonosíthatóság érdekében az itt meghatározott kóddal jelöltem FS-főkategória-alkategória alakban. E jelölésrendszernek megfelelően a nyílt forrás "Szabályozás" kategóriájába eső "Összetett licencelés" az FS-SZ-L, míg a "Fejlesztési folyamatok" kategória "Tervezéssel" kapcsolatos elemzése az FS-F-P kódot kapta (lásd: 0.5. ábra).



0.5. ábra: A nyílt forrás sajátosságainak rendszertana (szerkesztette a szerző)

A második fejezetben elemzett sajátosság kategóriák mindegyikénél meghatároztam a felmerülő sérülékenységeket valamint az azonosított javaslatokat. A sérülékenységeket 'S' míg a javaslatokat 'J' kezdetű kód jelöli, amelyet a kategória betűkódja, majd az egyediséget biztosító kétjegyű szám követ. Ennek megfelelően a Fejlesztés (F) kategóriában azonosított első lehetséges sérülékenységet az SF01, míg az első javaslatot JF01 kód jelöli.

A harmadik fejezet az azonosított FLOSS sajátosságok hatását elemzi a Létfontosságú Rendszerelelemek területén. Ez két lépésben történik, először a NIST Cyber Security Framework által definiált információbiztonsági irányelvek alapján vizsgálom a nyílt forrás sajátosságainak lehetésges hatásait az egyes lépésekre, majd az azonosított feladatokat összevetem a Létfontosságú Rendszerelemek információbiztonsági követelményeit definiáló hazai szabályozással.

Az egyértelmű azonosíthatóság érdekében a biztonsági irányelvek esetében megtartottam az eredeti (angol nyelvű) dokumentum kódjait, a megfelelő fejezetket ez a kód jelöli. Ennek megfelelően például a “Védelem” (Protection) alá tartozó “Adatbiztonság” (Data Security) elvével foglalkozó részt PR.DS kód jelöli az eredeti jelöléssel összhangban.

A bizonsági irányelvek és FLOSS sajátosságok összevetése során azonosított lehetséges védelmi intézkedéseket ‘V’ kód jelöli, amelyet a biztonsági irányelv kategória kódja, majd folyamatos számozás követ. Így tehát a Helyreállítás (Recovery) kategória alatt azonosított védelmi intézkedések VRC[N] alakú kódot kapnak, ahol N az 1-től induló folyamatos számozást jelenti.

1 Kutatás tárgya és irodalma

1.1 Létfontosságú Rendszerelemek és Információs Rendszerelemek

A Létfontosságú Rendszerelemek, régebbi nevén Kritikus Infrastruktúrák védelme korunk egyik legfontosabb biztonsági kihívása. Értekezésemben a létfontosságú rendszerelemek működési hátterét biztosító Létfontosságú Információs Rendszerelemekre gyakorolt hatásokkal foglalkozom.

1.1.1 Létfontosságú Rendszerelemek (Kritikus Infrastruktúrák)

A társadalom működésének megzavarását célzó támadások stratégiai célpontjai a kritikus infrastruktúrák. A kritikus infrastruktúrák működtetéséhez szükséges infokommunikációs rendszerek rendkívül csábító támadási felületet jelentenek, hiszen a támadó fél alacsony erőforrás és eszközigénnyel is jelentős károkat tud okozni, ráadásul a globális jellegből adódóan mindezt biztonságos távolságból teheti meg. A kritikus információs infrastruktúrák közötti szoros kapcsolat jelentősen megnöveli az információs társadalom sebezhetőségét. Az új típusú fenyegetések annál hangsúlyosabban vannak jelen, minél magasabb szintű e rendszerek integráltsága, komplexitása és egymástól való függősége. Ennek megfelelően egyre erősebb az igény a megfelelő szintű biztonság elérésére. [13]

A fenyegetés súlyát növeli, hogy a kritikus infrastruktúrák elleni információs támadások célja feltehetően nem gazdasági előnyszerzés hanem politikai vagy katonai célok elérése, a támadó pedig nem egyedül álló személy vagy kisebb csoport, hanem jelentős erőforrásokkal rendelkező nemzetállam.

Könnyen felismerhető, hogy ez a fajta katonai felfogás jól alkalmazható az egymással szoros függőségi

kapcsolatban álló kritikus infrastruktúrák esetén, így e rendszerek súlyponti elemeinek megfelelő védelme fontos nemzetbiztonsági kérdés.

A létfontosságú rendszerek és létesítmények azonosításáról, kijelöléséről és védelméről a 2012. évi CLXVI. törvény rendelkezik.

Létfontosságú rendszerelemnek minősül a törvény mellékletében meghatározott ágazatok valamelyikébe tartozó eszköz, létesítmény vagy rendszer olyan rendszereleme, amely "elengedhetetlen a létfontosságú társadalmi feladatok ellátásához - így különösen az egészségügyhöz, a lakosság személy- és vagyonbiztonságához, a gazdasági és szociális közszolgáltatások biztosításához -, és amelynek kiesése e feladatok folyamatos ellátásának hiánya miatt jelentős következményekkel járna." [23]

A törvény szerint európai létfontosságú rendszerelemnek minősül a törvény alapján kijelölt "olyan létfontosságú rendszerelem, amelynek kiesése jelentős hatással lenne - az ágazatokon átnyúló kölcsönös függőségből következő hatásokat is ideértve - legalább két EGT-államra".

Nemzeti létfontosságú rendszerelemnek minősül a törvény alapján kijelölt "olyan létfontosságú rendszerelem, amelynek kiesése a létfontosságú társadalmi feladatok folyamatos ellátásának hiánya miatt jelentős hatása lenne Magyarországon".

Létfontosságú rendszerelem védelmének "a létfontosságú rendszerelem funkciójának, folyamatos működésének és sértetlenségének biztosítását célzó, a fenyegetettség, a kockázat, a sebezhetőség enyhítésére vagy semlegesítésére irányuló" valamennyi tevékenységet nevezzük.

A kritikus infrastruktúrák védelmi szempontból legfontosabb jellemzői:

- kölcsönös függőség,
- hálózatszerűség,
- domino-elv hatás,
- működési sajátosság,
- kiterjedés-elhelyezkedés,
- informatikai háttér.

Az infrastruktúrák közötti fennálló függőség miatt jelentős negatív hatást lehet gyakorolni:

- az információs társadalom szervezett működési rendjére, minőségére, dinamikus egyensúlyára;

- vezetési rendszerének hatékonyságára, integritására;
- a vezetés struktúrájára, szervezettségi fokára;
- a belső és külső kommunikációra, a kapcsolati viszonyokra és
- az adott szervezet operatív vezethetőségére. [24]

A dominó elv értelmében tényleges valószínűsége van annak, hogy a kölcsönös függőség miatt egyetlen esemény láncreakciót indítson be, amelynek végül több infrastruktúra működését és rendelkezésre állását befolyásolja.

A kritikus infrastruktúra védelem három fő pilléren - megelőzés, felkészülés és ellenálló képesség – támaszkodva törekszik az infrastruktúrák biztonságos működését elősegítő intézkedések megtételére. A megelőzés első sorban a különböző leállások, meghibásodások kockázatának lehető legkisebb szintre történő csökkentésére irányul. A megelőzési tevékenységek közé soroljuk a veszélyeztető tényezők elemzését, a kockázatok feltérképezését és a legérzékenyebb pontok beazonosítását, amelyek alapján a szükséges védelmi szint meghatározható. A felkészülés feladata a tulajdonosok, üzemeltetők, felügyeleti szervek és központi államigazgatási szervek felkészítése, melynek célja, hogy az érintettek között aktív és hatékony kommunikáció, valamint eredményes és célravezető együttműködés alakuljon ki. Az ellenálló képesség megerősítéséhez három összetevő biztosítása szükséges:

- alternatívák kialakítása, a kieső rendszer mielőbbi pótlásának érdekében;
- visszaállító képesség, hogy az esemény után a lehető leghamarabb visszaálljon az eredeti állapot;
- végül a sebezhető pontok csökkentése, melynek segítségével az infrastruktúra ellenálló képessége megnövekszik tekintve, hogy a kritikusság mértékét immár kevesebb kockázati faktor határozza meg. [25]

1.1.1.1 Létfontosságú rendszerelemek fejlődése

A napjainkban is zajló cselekvési hullámot a 2001-ben elkövetett terrortámadások indították meg. A háttérbe szoruló hagyományos hadviselés helyét az új, nehezen azonosítható fenyegetések vették át, amelyek egyre inkább irányulnak kifejezetten a lakosság elrettentésére és károkozásra, mint a fegyveres erők ellen.

A NATO Felsőszintű Polgári Veszélyhelyzet Tervezési Bizottsága 2001-ben kezdte meg a kritikus inf-

rastruktúra védelemmel kapcsolatos irányelvek kidolgozását, amelynek keretében a 2003-ban elfogadott irányelvben a megalkotta saját, tevékenységéhez illeszkedő kritikus infrastruktúra fogalmát, amely "azokat a létesítményeket, szolgáltatásokat és információs rendszereket jelenti, amelyek olyan létfontosságúak a nemzetek számára, hogy működésképtelenné válásuknak vagy megsemmisülésüknek gyengítő hatása lenne a nemzet biztonságára, a nemzetgazdaságra, a közegészségre, a közbiztonságra és a kormány hatékony működésére"[1] [25].

Az események az Európai Uniót is cselekvésre késztették, felvetve az egységes, stratégia alapú, közösségi infrastruktúrák védelmére irányuló jogi szabályozás igényét. Az Európai Unión belül 2004-től kezdve több, a Létfontosságú Rendszerelemekkel kapcsolatos dokumentum jelent meg:

- Communication from the Commission to the Council and the European Parliament: Preparedness and consequence management in the fight against terrorism, COM (2004) 701 final, Bruxelles, 20 October 2004;
- Communication from the Commission to the Council and the European Parliament: Critical Infrastructure Protection in the fight against terrorism, COM (2004) 702 final, Bruxelles, 20 October 2004;
- Green paper on a European programme for critical infrastructure protection (presented by the Commission) COM (2005) 576 final, Bruxelles, 17 November 2005.

Ezeket követte a 2008/114/EC direktíva megjelenése[2], amely értelmében a legfőbb és végső felelősség a tagállamokat és az infrastruktúra üzemeltetőjét terheli [26].

Az Európai Unió kritikus infrastruktúra fogalma részletesebb, kifejezetten az Unió egységére irányul, de mégis általánosságban fogalmaz. A Zöld Könyv definíciója szerint "azok a fizikai eszközök, szolgáltatások, információs technológiai létesítmények, hálózatok és vagyontárgyak" tekinthetők kritikus infrastruktúrának, "melyek megrongálása vagy elpusztítása súlyos hatással lenne az európaiak egészségére, békéjére, biztonságára, vagy gazdasági jólétére illetve az EU és a tagállamok kormányainak hatékony működésére"[3] [25]

[1] A NATO Polgári Védelmi Bizottsága által megfogalmazott kritikus infrastruktúra védelmi koncepció alapján (Critical Infrastructure Protection Concept Paper EAPC(SCEPC)D(2003)15).

[2] Council Directive 2008/114/EC of 8 December 2008 on the identification and designation of European critical infrastructures and the assessment of the need to improve their protection.

[3] (Az Európai Unió Zöld Könyve alapján (Green Paper on European Programme for Critical Infrastructure Protection COM(2005) 576 final)

A Zöld Könyv elsődleges célja, hogy felhívja a figyelmet egy-egy terület meghatározó és megválaszolatlan kérdéseire, illetve aktív együttműködésre ösztönözze az egyes szektorok képviselőit, mint a létfontosságú infrastruktúrák védelmének kulcsszereplőit. Központi eleme a szubszidiaritás[4] került, amely szerint a létfontosságú rendszerelemek védelme elsősorban nemzeti hatáskör, vagyis a felelősség első sorban a tagállamokat, tulajdonosokat és üzemeltetőket terheli.

A későbbi irányelv kidolgozásához a Zöld Könyv három védelmi stratégiát kínált, amelyekre a védelmi tevékenység felépíthető. Egy összetett megközelítés, amely a szándékos, ártó jelleg támadásokkal és a természeti katasztrófák veszélyeivel egyaránt számol, de a terrorizmust nem kezeli kiemelt kihívásként. Egy komplex és rugalmas megközelítést, amely tekintettel van az egyéb támadásokból származó fenyegetésekre és a természeti katasztrófák okozta veszélyekre, de középpontjában az ártó szándékú cselekmények vagyis a terrorizmus áll. Végül pedig egy kifejezetten a terrorizmusra összpontosító megközelítést, amely nem tekint prioritásnak más egyéb veszélyeztető tényezőt. [27]

Meghatározásra került a közösség érdekei szerint elsődleges és magasabb szintű létfontosságú rendszerelemek, azaz európai létfontosságú rendszerelemek[5] fogalma. A másodlagos, de a tagállamok szempontjából kiemelt jelentőséggel bíró csoport a nemzeti létfontosságú rendszerelemek[6] nevet kapta, amelyek létesítésük szerint egy adott tagállam területén találhatóak, de sérülésük, leállásuk, megsemmisülésük esetén hatásuk csak az ország határain belül érezhető. [25] Fentieken túl a Zöld Könyv felhívta a figyelmet az Unió területén kívül található olyan infrastruktúrákra, amelyek esetleges üzemzavara, vagy megsemmisülése hatással lehet az Európai Unió tagállamaira, azok infrastruktúráira, működésére.

Az európai kritikus infrastruktúrák azonosításáról és kijelöléséről, valamint védelmük javítása szükségességének értékeléséről szóló 2008/114/EK Irányelvet 2008. december 8-án fogadták el. Az Irányelv a Zöld Könyvvel, az európai programmal és az ágazati specifikumokkal összhangban, határozta meg a létfontosságú rendszerelemek azonosítására és kijelölésére vonatkozó eljárások, eszközök és elvek halmazát. Az Irányelv első és legfontosabb tétele, hogy a kihirdetéstől számított két éven belül a megvalósításhoz szükséges intézkedéseket a tagállamoknak végre kellett hajtaniuk. A 2011-ben megjelent EU belbiztonsági Stratégia tovább erősítette a kritikus infrastruktúrák védelmének szükségességét és napirenden tartását,

[4] A döntéseket azon a lehető legalacsonyabb szinten kell meghozni, ahol az optimális informáltság, a döntési felelősség és a döntések hatásainak következményei a legjobban átláthatók és érvényesíthetők.
[5] European Critical Infrastructure, ECI
[6] National Critical Infrastructure, NCI

különös tekintettel a modern technológiák által biztosított lehetőségeket is kiaknázó szervezett bűnözésre. [27]

1.1.1.2 Magyarországi tevékenységek

NATO tagságunk révén betekintést nyertünk a kritikus infrastruktúra védelmi kezdeményezésekbe, 2001-2002-ben országos adatgyűjtés keretében felmérték az infrastruktúrák helyzetét, amely alapján megállapításra került, hogy kormányzati szint döntésekre van szükség ahhoz, hogy az infrastruktúrák védelmével kapcsolatos tevékenység megfelelő háttere biztosítva legyen. [27]

A létfontosságú rendszerelemek nemzetközi védelmi programjának kidolgozásában Magyarország már teljes jogú tagállamként vett részt. A 2006 decemberében elfogadott irányelv-tervezet alapján megkezdődhetett az érdemi munka hazánkban is, 2007-ben a tárcák és országos hatáskörű szervek együttműködésével vette kezdetét a magyar nemzeti védelmi program kidolgozása. Megszületett a magyar Zöld könyv, mely már érvényesítette a kritikus infrastruktúrák ágazati szintű besorolását. A Zöld könyv megállapítása szerint: "az infrastruktúrák folyamatos működése, kockázati tényezőkkel szembeni ellenálló képességének növelése a lakosság, az infrastruktúra tulajdonosok, üzemeltetők, valamint a gazdaság szereplői és az állam számára egyaránt kiemelt fontossággal bír, a biztonságos működést elősegítő környezet és intézkedések ezért értéket képviselnek."

A 2080/2008. (VI. 30.) kormányhatározat a Kritikus Infrastruktúra Védelem Nemzeti Programjáról kihirdette a további konzultációk és folyamatok alapjául szolgáló zöld könyvet. Elrendelte az ágazati konzultációk lefolytatását, amelyhez ágazatonként minisztériumot, vagy országos hatáskörű szervet rendelt felelősként; valamint szabályozási koncepció kidolgozását írta elő [25]. A határozat jogszabályi formába ültette át a Zöld könyvet, így megindulhatott a pár- beszéd a potenciális üzemeltetők, kritikus infrastruktúra tulajdonosok és az érintett hatóságok között. A következő lépés a 1249/2010. (XI.19.) Korm. határozat kiadása volt, mely lényegében az irányadó EU irányelv implementációja érdekében végrehajtandó kormányzati feladatok katalógusát adta meg, amit hamarosan követet az új katasztrófavédelmi norma, a katasztrófavédelemről és a hozzá kapcsolódó egyes törvények módosításáról szóló 2011. évi CXXVIII. törvény és végrehajtási rendelete, a 234/2011 (XI. 10.) Korm. rendelet kiadása. A jogszabály a BM Országos Katasztrófavédelmi Főigazgatóság (a továbbiakban: OKF) szervezetén belül lérehozta az iparbiztonsági

főfelügyelőséget, amely a súlyos balesetek elleni védekezésért, a veszélyesáru-szállítás felügyeletéért és a kritikus infrastruktúrák védelméért felelős [1].

A létfontosságú rendszerek és létesítmények azonosításáról, kijelöléséről és védelméről szóló 2012. évi CLXVI. törvényt (Lrtv.) az Országgyűlés 2012. november 12-én fogadta el. A törvény célja a létfontosságú rendszerelemek azonosítása, illetve a kijelölést követően a megfelelő szint védelem biztosítása. Az Lrtv. az értelmező rendelkezésekben található alapvető fogalmak meghatározásán túl, körvonalazza a nemzeti és az európai kritikus infrastruktúrák kijelölésének rendjét, rendelkezik az üzemeltetői biztonsági terv készítésének kötelezettségéről, a biztonsági összekötő személy kijelöléséről, a nyilvántartás és ellenőrzés szabályairól, valamint a szankcionálásról. Külön rendelkezéseket tartalmaz az energia ágazat vonatkozásában, amelyeket az ágazati kormányrendelettel együtt kell értelmezni.

A keretszabályozás markáns eleme lett a Lrtv. végrehajtási rendeletének tekinthető 65/2013. (III. 8.) kormányrendelet, amely részletesen szabályozza az azonosítási és kijelölési eljárás általános folyamatát, beleértve az azonosítási jelentést, az ágazati kijelölő és javaslattevő hatóság szerepét, és a szakhatósági állásfoglalást. Külön rendelkezik az európai létfontosságú rendszerelemek kijelölési eljárásáról, a biztonsági összekötő személy általános követelményeiről, amelyeket az ágazati kormányrendeletek további, szakmai feltételekkel egészítenek ki. A rendelet – a honvédelmi létfontosságú rendszerelemek kivételével – OKF-et, jelöli ki az európai és nemzeti létfontosságú rendszerelemek nyilvántartó hatóságaként. [27]

1.1.2 Léfontosságú Információs Rendszerelemek

A 2013. évi L. törvény megfogalmazása szerint “létfontosságú információs rendszerelem az európai vagy nemzeti létfontosságú rendszerelemmé a létfontosságú rendszerek és létesítmények azonosításáról, kijelöléséről és védelméről szóló törvény alapján kijelölt létfontosságú rendszerelemek azon elektronikus információs létesítményei, eszközei vagy szolgáltatásai, amelyek működésképtelenné válása vagy megsemmisülése az európai vagy nemzeti létfontosságú rendszerelemmé kijelölt rendszerelemeket vagy azok részeit elérhetetlenné tenné, vagy működőképességüket jelentősen csökkentené”.

A törvény végrehajtási rendeletének definíciója szerint a létfontosságú információs rendszerek és létesítmények “a társadalom olyan hálózatszerű, fizikai vagy virtuális rendszerei, eszközei és módszerei, amelyek az információ folyamatos biztosítása és az informatikai feltételek üzemfolytonosságának szükségességéből

adódóan önmagukban létfontosságú rendszerelemek, vagy más azonosított létfontosságú rendszerelemek működéséhez nélkülözhetetlenek.”

A jogszabályi megfogalmazás egyértelművé teszi, hogy egy kritikus információs infrastruktúra lehet önmagában is kritikus, illetve teljesíti a kritikusság kritériumát a többi kritikus infrastruktúra számára nyújtott nélkülözhetetlen infokommunikációs szolgáltatás által is. A fenti definíció és a törvény ágazati besorolása alapján a kritikus információs infrastruktúrák alatt az alábbiakat értjük:

- az energiaellátó rendszerek rendszerirányító infokommunikációs hálózatait;
- a közlekedésszervezés és -irányítás infokommunikációs hálózatait;
- az agrárgazdaságot szabályzó, irányító infokommunikációs hálózatokat;
- az egészségügyi rendszer infokommunikációs hálózatait;
- a pénzügyi szektor infokommunikációs hálózatait;
- az ipari termelést irányító infokommunikációs hálózatokat;
- az infokommunikációs technológiákat;
- a vízellátást szabályzó infokommunikációs hálózatokat;
- a jogrend és a kormányzat infokommunikációs hálózatait;
- a közbiztonság és védelmi szféra infokommunikációs hálózatait. [13]

A létfontosságú információs rendszerelemek a létfontosságú rendszerelemek sajátos veszélyei mellett további hét, jellemzően aszimmetrikus fenyegetésnek vannak kitéve:

- kiberbűnözés;
- kiberterrorizmus;
- hacktivista mozgalmak;
- programozási/paraméterezési és üzemeltetési hibák.

Ezek jelentős károkat eredményezhetnek, negatívan befolyásolhatják az üzem- és termelés folytonosságot, a hatékony vállalati működést, és az infrastruktúrák összekapcsolódásából eredő hatások folytán ágazati szintű problémákat okozhatnak. [1]

1.1.3 LRE, LIRE és FLOSS kapcsolata

Mint láthattuk, a Létfontosságú Információs Rendszerelemek védelme kiemelt feladat amelyet törvényi szabályozás is előír. Tekintve, hogy a LIRE rendszere igen összetett is lehet, könnyen elképzelhető, hogy közvetve vagy közvetlenül FLOSS komponenseket tartalmaz. A nyílt fejlesztési modellből származó FLOSS termékek biztonságossága illetve sérülékenysége éles vitákat váltott ki a szakemberek körében, ráadásul ezek a megoldások máshol nem tapasztalható egyedi sajátosságokkal rendelkeznek. Jogosan merül fel a kérdés, hogy a FLOSS komponensek egyedi sajátosságai – amennyiben a rendszer valóban tartalmaz ilyeneket – mennyiben befolyásolják a biztonságot, illetve milyen intézkedések hozhatók a kockázatok csökkentése érdekében.

1.2 Nyílt forráskód értelmezése

A kutatás témájául szolgáló nyílt forráskód meghatározása nem egyértelmű, hiszen ha megkérdezünk egy menedzsert, egy fejlesztőt és egy biztonsági szakembert, szinte bizonyos, hogy némiképp hasonló de alapvetően háromféle választ fogunk kapni. A fogalom mindenkinek kicsit mást jelent, amit tovább bonyolít a számtalan különféle elnevezés amelyek között gyakoriak a fogalmi átfedések.

Nevezik nyílt forrásnak, szabad szoftvernek, az angol nyelvben elterjedt a open source software, libre software, free software kifejezés, a F/LOSS és OSSD betűszavak, illetve találkozhatunk a F/OSS, F/OSSD, FOSSD alakokkal is. Érthetnek alatta jogi fogalmat, filozófiát, közösséget, fejlesztési módszertant, terméket vagy éppen a konkrét forráskódot.

Kutatásomban a nyílt forrásra mint jelenségre koncentráltam, tehát nem az egyes ingyenesen elérhető szoftvertermékeket értem alatta, hanem azt az összetett társadalmi jelenséget és módszertant, ami által ezek a termékek és licencek létrejöttek, beleértve azok minden aspektusát. Ezt a jelenséget a kutatás során Nyílt Fejlesztési Modellnek nevezem. A Nyílt Fejlesztési Modell által létrehozott, licencel rendelkező szoftvertermékekre Szabad Szoftverként fogok hivatkozni, hogy egyértelműen megkülönböztessem őket az egyéb, nem termék jellegű forráskódtól és információtól. Rövidítésként viszont a kevéssé szerencsés magyar rövidítés helyett a nemzetközi gyakorlattal jobb összhangban álló FLOSS betűszót használom.

Nyílt forráskód alatt egy bővebb halmazt értek, amely magába foglalja a Szabad Szoftverek forráskódját, azok különféle kiadott és nem kiadott valamint fejlesztői verzióit, a verziókhoz tartozó metainformációkat, foltokat, a portálokon, blogokon, levelezőlistákon elérhető kóddarabkákat, javításokat, egyszóval minden olyan publikált forráskódszerű fejlesztői információt, amelyeket a licencek már esetleg nem fednek le, de általában mégis könnyen hozzáférhetőek.

A nyílt fejlesztési modell célját és motivációit a létrehozott termék, a Szabad Szoftver definícióján keresztül lehet a legkönnyebben megérteni. A következő fejezetben ezt fogom tehát bemutatni.

1.2.1 Szabad Szoftver

A nehezen átlátható helyzetet jól jellemzi, hogy a fogalmat szabványosítani kívánó két szervezet a megnevezésben sem tudott közös nevezőre jutni [28]. A Richard Stallman ideológiai örökségét követő Free Software Foundation (FSF) a Free Software név mellett van és a szabadságjogokra helyezi a hangsúlyt, a valamivel üzletiesebb filozófiát követő Open Source Initiative (OSI) ellenben az Open Source megnevezést támogatja és inkább a praktikus és jogi aspektusra koncentrál.

Valójában a két fogalom közt nincs jelentős különbség, a nyílt forrásnak vagy szabad szoftvernek teljesíteni kell a következő 4 szabadsági jogot:

0. jogot arra, hogy futtassák a programot, bármilyen céllal;
1. jogot arra, hogy tanulmányozzák a program működését, és azt a szükségleteikhez igazíthassák. Ennek előfeltétele a forráskód elérhetősége.
2. jogot arra, hogy másolatokat tegyenek közzé a felebarátaik segítése érdekében;
3. jogot arra, hogy tökéletesítsék a programot, és a tökéletesített változatot közzétegyék, hogy az egész közösség élvezhesse annak előnyeit. Ennek előfeltétele a forráskód elérhetősége.

Az OSI megfogalmazásában, lényegében ugyanez 10 pontba szedve (és némiképp lerövidítve) így hangzik:

1. *Szabad terjesztés.* A licenc nem korlátozhat semmilyen felet aki a szoftvert egy nagyobb szoftverdisztribúció részeként el szeretné adni, és nem kérhet ezért díjat sem.

2. *Forráskód.* A programnak tartalmaznia kell a forráskódot, és engedélyeznie kell a bináris és forráskód formájú terjesztést. Ha a terméket forráskód nélkül árusítják, a forrás beszerezhetőségét egyér-

telműen meghatározott helyen, józan másolási díj ellenében biztosítani kell, lehetőleg az letölthető formában.

3. *Származtatott munkák.* A licencnek engedélyeznie kell a módosítást és a származtatott munkák létrehozását, és meg kell engednie, hogy azt azonos feltételek mellett terjesszék.

4. *Szerző forráskódjának megőrzése.* A licenc csak akkor tilthatja az forráskód módosított formájának terjesztését, ha a licenc megengedi a folt (patch) állományok terjesztését, amivel fordításkor az eredeti forrás módosítható. A licencnek explicit engedélyeznie kell az ilyen forrásból létrehozott szoftver terjesztését.

5. *Személyek és csoportok diszkriminációjának tilalma.* A licenc nem diszkriminálhat személyeket vagy csoportokat

6. *Területek diszkriminációjának tilalma.* A licenc nem tilthatja meg hogy a szoftvert egy adott területen használják (pl. üzleti felhasználás, vagy genetikai kutatások területe)

7. *Licenc terjesztése.* A programhoz kötődő jogok legyenek érvényesek mindenkire aki a terjesztés során azt megkapta, anélkül hogy egyéb licencek kötelezettségeinek meg kellene felelniük.

8. *A licenc nem lehet termékspecifikus.* A programra vonatkozó jogok nem függhetnek attól hogy a program egy nagyobb disztribúció része-e. Amennyiben a programot kiemelik a disztribúcióból és a program licence alapján terjesztik, azokat akik megkapták az eredeti programmal azonos jogok illetik meg.

9. *A licenc nem korlátozhat más programokat.* A licenc nem korlátozhat vele együtt terjesztett más programokat, nem mondhatja például, hogy a vele azonos médiumon lévő programok is nyílt forrásúak legyenek.

10. *A licenc legyen technológia-független.* A licenc nem tehet kitételeket semmilyen technológiára vagy interfész stílusra.

Mint látható az OSI verzió inkább a kiskapuk explicit bezárására koncentrál, de elveit tekintve nagyon hasonló az FSF verziójához. Jelenleg több mint 80 OSI kompatibilis licenc létezik. Legismertebb és egyben leggyakrabban használt ezek közül az Apache, BSD, a GNU General Public License (GPL), GNU Library vagy “Lesser” General Public License (LGPL), az MIT és a Mozilla.

Néhány licenc nagyobb szabadságot ad mint mások, általában két nagy csoportot az kötött és engedékeny nyílt forrású licenceket különböztetjük meg. Részletesebben 2.7.2 fejezetben foglalkozom a licencelés kérdésével.

1.2.2 Nyílt Fejlesztési Modell fajtái

A kezdetek kezdetén valójában minden kód nyílt forrású volt, hiszen a korai fejlesztők kutatók voltak, akik nem terveztek gazdasági hasznot húzni az általuk írt kódból, szabadon cserélték és terjesztették azt. Helyesebb lenne tehát úgy fogalmazni, hogy minden szoftver őse a nyílt forráskódú szoftver, azaz ez volt a Szoftver, és csak később a szoftveripar üzletiesedésével jelentek meg a licencelt vagyis jogokhoz kötött, *korlátozott szoftverek*.

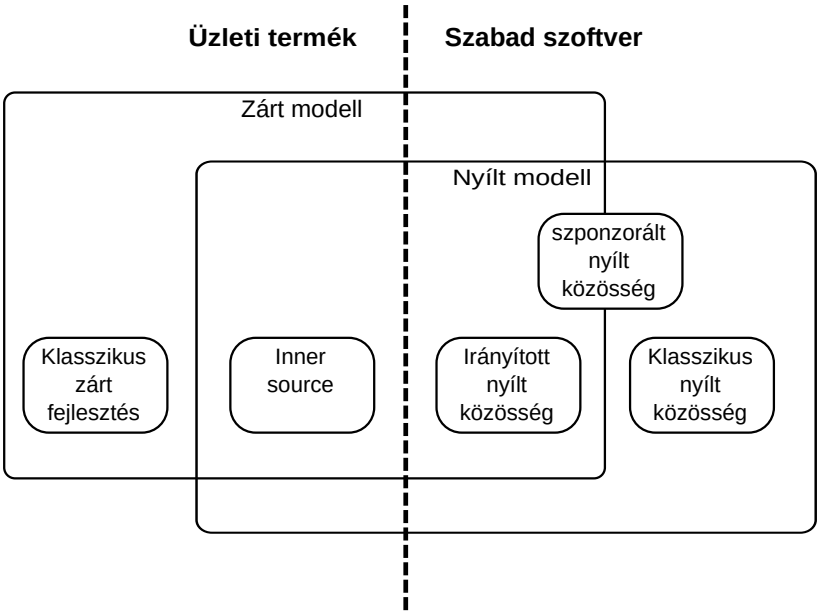
A Nyílt Modelltől való egyértelmű megkülönböztetés érdekében az ilyen termékek létrehozásához vezető módszertant és gondolkodásmódot *Zárt Modellnek*, a létrehozott terméket pedig *üzleti szoftvernek* fogom nevezni.

A számítástechnika hőskora óta mindkét modell sokat változott, mindkét oldal egyes csoportjai integráltak bizonyos számukra előnyös elemeket a másik módszertanából, miközben más csoportok megtartották a korai "tiszta" irányzatot. Így napjainkban mindkét modell számos változata létezik párhuzamosan egymás mellett.

A Nyílt modell alapja a *Közösség*, amely a fejlesztők és felhasználók egymással folyamatosan kommunikáló, és egymást részben átfedő halmaza.

Az eredeti, üzleti módszereket nem használó, sőt gyakran elutasító közösséget *klasszikus nyílt fejlesztői közösségnek* vagy röviden klasszikus nyílt közösségnek nevezem. A klasszikus nyílt közösségben csak a felhasználók között találunk cégeket, a fejlesztést teljes egészében a közösség végzi, pénzbeli ellenszolgáltatást a fejlesztők nem kapnak.

A nyílt modell előretörésével a cégek egyre inkább szerepet kívántak vállalni a fejlesztésekben, amit először támogatások, később pedig fizetett fejlesztők formájába tehettek meg, amivel megszületett a *szponzorált nyílt közösség*. A szponzorált nyílt közösséget – amelyben a cégek is fejlesztő szerepet töltenek be – a nemzetközi irodalomban általában Sponsored Open Source Community néven találhatjuk meg [29].



1.1. ábra: Nyílt és Zárt fejlesztési modell, közösségek és termék összefüggései (szerkesztette a szerző)

A nyílt közösség harmadik alaptípusa, amikor a közösség kezdetektől fogva egy vagy több irányító szerepet betöltő cég körül szerveződik vagy egy piaci szereplő idővel átveszi a projekt irányítását. Ez történhet úgy, hogy a cég megnyitja valamilyen korábban zárt modellben készült szoftverének forráskódját, illetve napjainkban egyre gyakrabban úgy is, hogy eleve a kezdetektől nyílt modellben szeretne fejleszteni, miközben az irányítást biztos kézben tartja. Az ilyen közösségeket alapvetően az ipar irányítja, ezért ezeket *irányított nyílt közösségnek* nevezem.

/ ## A nyílt modell hatáspontjai{#sec:FLOSSHP}

A FLOSS hatásainak elemzéséhez ismernünk kell azokat az útvonalakat, amelyeken keresztül a létfontosságú információs rendszerelemek kapcsolatba kerülhetnek a FLOSS komponensekkel. Ebben a fejezetben ezeket a hatásmechanizmusokat, tulajdonképpen az egyes rendszerelemek FLOSS vonatkozású beszállítói láncát elemzem.

Bizonyos direkt hatáspontok – mint például a nyílt forrású operációs rendszer vagy dobozos termék használata – azonnal egyértelműek, míg más hatások közvetett, rejtett módon érvényesülnek, ezért nagyon fontos, hogy egy esetleges kockázatelemzés során ezeket is figyelembe vegyük.

Az létfontosságú rendszerelemre alapvetően háromféle módon lehet hatni:

- saját létfontosságú információs rendszerén keresztül;

- más, vele függőségi kapcsolatban álló LRE[7]-re gyakorolt hatás által,
- illetve beszállítói láncának támadásával.

Az utóbbi két lehetőséget természetesen figyelembe kell venni egy esetleges kockázatelemzés során, a kutatás tárgyát képező esetben viszont ezek a hatások a létfontosságú információs rendszerelemre gyakorolt hatáspontok alhalmazát képezik, ezért együtt kezelem őket a LIRE hatáspontjaival.

A LIRE[8] felhasználhat közvetlenül FLOSS elemeket vagy FLOSS elemeket tartalmazó rendszereket és szolgáltatásokat, illetve kapcsolatban állhat olyan más LIRE-el amelytől saját működése nagy mértékben függ. Ez utóbbi csoportba tartoznak például az infokommunikációs hálózatok. A kutatás során a direkt hatásokat vizsgálom, hiszen a kölcsönhatásban érintett LIRE rendszerre azonos felételek vonatkoznak, így értelemszerűen az összes lehetséges hatáspontot a hatást gyakorló LIRE esetében is vizsgálni kell.

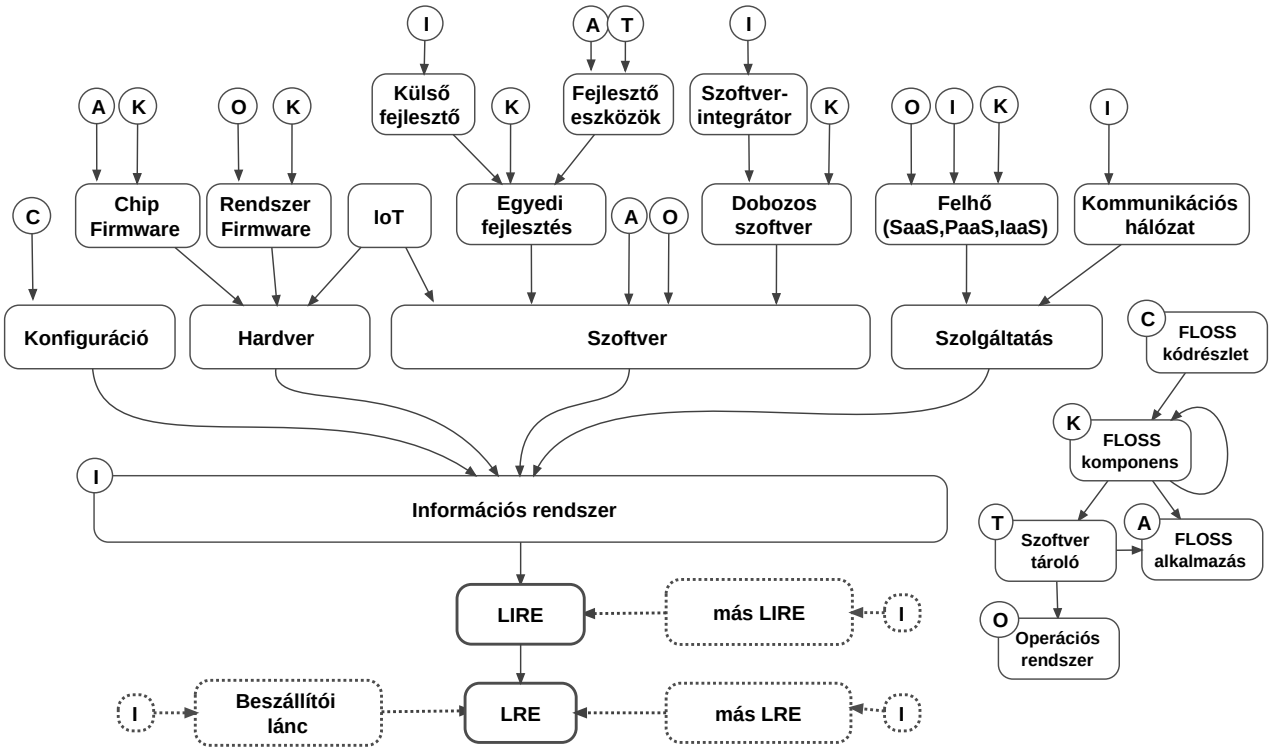
A LIRE felhasználhat:

- FLOSS komponenseket tartalmazó hardvereket,
- FLOSS vagy FLOSS komponenseket tartalmazó szoftvereket,
- szolgáltatásokat,
- illetve nyílt forrásból származó konfigurációs elemeket.

A lehetséges hatáspontokat a 1.2. ábra foglalja össze. Mint látható, a hatások komplex, többszörösen áttételes módon is érvényesülhetnek. A FLOSS komponensek (K) felhasználhatnak más FLOSS komponenseket illetve kódrészleteket (C), a komponensek közvetlenül vagy szoftvertárolók közvetítésével bekerülhetnek a FLOSS alkalmazásokba (A) illetve az operációs rendszerbe (O). Természetesen maga az alkalmazás illetve az operációs rendszer is lehet nyílt forrású, de ez nem feltétlen követelmény.

A szervezet információs rendszerének egyik lehetséges hatáspontja a hardver. Nyílt forrású lehet vagy nyílt forrásból származhat a felhasznált hardver firmware kódja vagy valamely hardver komponensének firmware kódja. A firmware maga jelenleg döntő részben zárt, ugyanis a gyártó saját versenyelőnyét megőrizendő igyekszik a hardvert vezérlő szoftver komponenseket titokban tartani. Ezt a jelenséget erősíti, hogy számos gyártó a hardverkomponenseket kevésbé szabályozott de olcsóbb előállítást ígérő piacokon készítteti el, ugyanakkor az illegális másolást csak úgy tudják kivédeni, ha a hardver működtetését végző

[7]Létfontosságú Rendszerelemek
[8]Létfontosságú Információs Rendszerelem



1.2. ábra: FLOSS hatáspontjai a LRE rendszerekre (szerkesztette a szerző)

firmware-t üzleti titokként őrzik meg. Ez a jelenség azonban szembemegy a biztonságot jelentősen befolyásoló átláthatóság követelményeivel, hiszen az ismeretlen firmware tetszőleges kódot tartalmazhat, ráadásul közvetlenül, az operációs rendszer megkerülésével éri el a hardvert, így a forrás ismerete nélkül ellenőrzése rendkívül időigényes és nehézkes. Elképzelhető tehát, hogy a jelenleg már létező nyílt firmwarek száma a biztonsági követelmények szigorodásával párhuzamosan a jövőben növekedni fog. Hasonlóképpen növelheti a nyílt firmware elterjedését, ha a keleti piacok másolásvédelmi szabályozása javul vagy ha a termelés a robotika fejlődésével más területre helyeződik át.

A második lehetséges hatáspont a szoftverelemeken keresztül érvényesül. A felhasznált szoftver lehet maga is FLOSS illetve futhat nyílt forrású operációs rendszeren, a dobozos termék felhasználhat FLOSS komponenseket illetve az azt előállító integrátor infrastruktúrája egyaránt használhat nyílt szoftvert. Saját belső fejlesztés esetén mind a felhasznált komponensek mind a fejlesztő környezet irányából számíthatunk FLOSS hatásokra. Ha szervezet külső fél segítségét veszi igénybe a belső fejlesztésekhez, a külső fejlesztő információs rendszerének valamennyi FLOSS hatáspontja elvben hatással lehet a szervezet biztonsági szintjére is.

Fontos megemlíteni, hogy az integrátor nem tehető felelőssé az általa felhasznált FLOSS termék által

okozott károkért [30] hacsak a felhasználó nem fizet külön egy ilyen, biztosítás jellegű szolgáltatásért. Ennek következtében az integrátorokon keresztül beszivárgó FLOSS elemek problémája hatványozottan érvényesül.

A jelenleg feltörekvőben lévő IoT technológia szintén hatással lehet a biztonságra. Ugyan a jelenleg használt IoT komponensek forráskódja többnyire nem nyílt, szinte kivétel nélkül nagy mennyiségű nyílt forrásból származó kódot tartalmaznak és egyre nagyobb a nyomás az átláthatóság növelésére is. Amennyiben a szervezet IoT eszközöket is használ, a FLOSS hatáspontjai a hardver és szoftver lehetőségeinek unióját teszik ki. Logikus és egyszerű lenne az szervezet IoT eszközeit az információs rendszer hardver elemeivel azonos módon kezelni, azonban ebben az esetben a szabályozás nagyságrendekkel nehezebb, a hardver nem kis számú, ellenőrzött szállítótól származik és az eszközöket sem feltétlenül jól képzett szakemberek használják. A lassan de biztosan terjedő, munkavállalóktól nehezen elválasztható számtalan apró kommunikációs eszköz, okosóra, egészségügyi berendezés, smart-ruházat kommunikációjának ellenőrizhetetlen volta jelentős problémát okozhat a közeljövőben. Ezeknek az eszközöknek az átláthatósága, közös ellenőrzése fontos biztonsági követelmény és egyben veszélyforrás is lehet a jövőben.

A szervezet által használt szolgáltatások – elsősorban a más szolgáltatások alapját képző felhőszolgáltatások és a kapcsolattartásért felelős kommunikációs hálózat – szintén a FLOSS belépési pontjaiként szolgálhatnak. Amennyiben ezen szolgáltatások zavara vagy leállása a létfontosságú rendszerelem alapvető funkcióit veszélyeztetné, a 1.1.2 fejezetben leírt LIRE definíciója alapján maga is létfontosságú rendszerelemnek tekintendő, következésképpen a LIRE esetében azonosított valamennyi hatáspont éppúgy vonatkozik rá.

A FLOSS hatása érvényesülhet technikai úton – például sérülékenység, hiba, hátsó kapu formájában – illetve jogi szinten, a felhaszált elemek licencelési problémái által. A jogi megközelítés valójában a 1.2. ábrán bemutatottnál is valamivel összetettebb, hiszen nyílt licencekkel rendelkezhetnek az alábbi komponensek:

- végrehajtható állományok,
- szoftver szolgáltatások (services),
- connector-ok,
- kapcsolódási módszerek,
- illetve maga a konfigurált rendszer vagy alrendszer architektúra. [31]

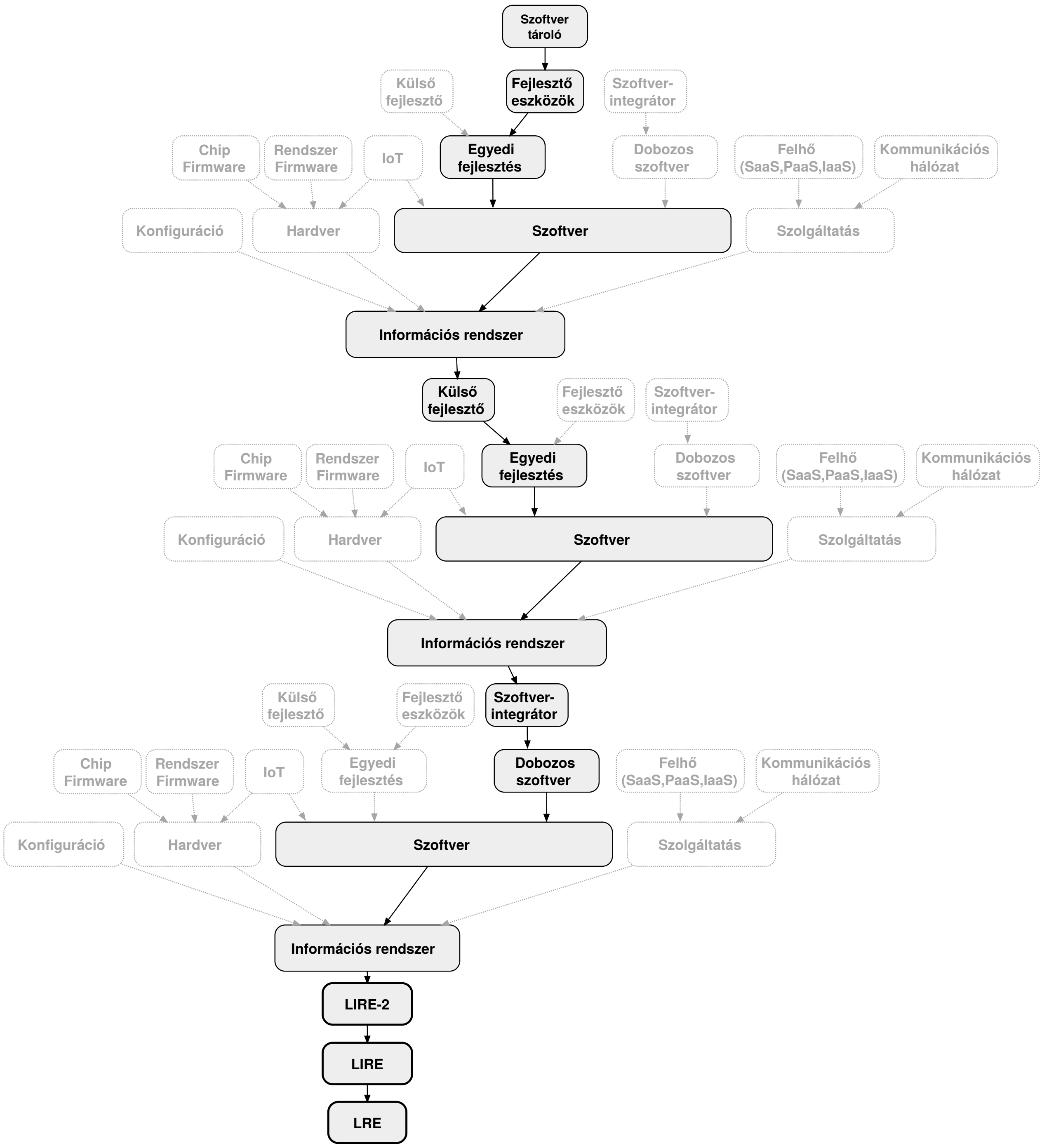
Ismert, hogy a fejlesztők gyakran keresnek valamilyen célzott vagy általános keresőprogrammal kódrészleteket. Ez általában egy FLOSS program részlete [32], következésképpen a nyílt forrás hatása teljes mértékben rejtetten is megjelenhet, úgy, hogy annak semmi nyoma nincs, hacsak az fejlesztő nem emlékszik honnan másolta a kódot. A hatás ilyen esetekben szinte kimutathatatlan. A kódmásolás problémája megjelenthet a konfigurációs állományokban – ami rendkívül gyakori jelenség – és magában a forráskódban is. Tekintve, hogy a forráskódot több ország joggyakorlatában szerzői jog védi, szabad másolása korlátozott, így ez az eset elméletileg ritkábban fordul elő. A gyakorlat azonban ettől saját tapasztalataim szerint is eltérő, a könnyen hozzáférhető nehezen beazonosítható jogállású kódrészletek csábítása igen erős, a szoftverfejlesztők előszeretettel használják azokat [33]. Biztonsági szempontból különösen aggályos, ha ezeket a kódrészleteket – bízva azok szerzőiben – nem nézik át és tesztelik megfelelőképpen.

A jelenlegi fejlesztői gyakorlatban külső szoftver komponensek felhasználása széles körben elterjed. A teljesen egyedi fejlesztést csak nagyon kevés gyártó engedheti meg magának – akkor is csak általában egy szűk, jól definiált területen, mint például a útvonalválasztók szoftverei – a szoftverek összetettsége és az elvárt funkcionalitás egyaránt jelentősen megnőtt, egyszerűen nincs idő minden komponens egyedi kifejlesztésére. A felhasznált komponensek származhatnak zárt forrásból, de a kifejezetten jó minőségű, nyílt – és ingyenesen hozzáférhető – komponensek elterjedésével a versenyképesség megtartása érdekében egyre több gyártó hajlandó, illetve kénytelen nagy mennyiségű nyílt komponenssel dolgozni. A nyílt komponensek egyik fő jellegzetessége, hogy előszeretettel használnak más nyílt komponenseket, ezáltal összetett függőségi hálózatot hozva létre. A jelenséggel bővebben a FS-F-P fejezetben foglalkozom.

A szoftveres komponenseket gyakran tárolják valamilyen szoftvertároló segítségével, következésképpen ezek megbízhatósága az eredmény biztonsági szintjére is kihatással lesz.

Tekintve, hogy a teljes szoftver-előállítási lánc valamilyen adott szoftverkörnyezetben fut, az ott alkalmazott operációs rendszer és segédeszközök (pl. build automation framework) biztonsági szintje szintén hatással lesz a felhasználásra kerülő rendszer biztonságára. Egy kompromittált rendszeren futó fordítóprogramok és fejlesztői eszközök olyan sérülékenységeket vezethetnek be a binárisokba, amelyek felderítése nagyságrendekkel nehezebb mint a forrás elemzése, így ezeknek a teszteknek az elvégzése csak a legritkább esetben gazdaságos.

Mint látható, a szoftveripar jelen fejlettségi szintje mellett a FLOSS számtalan, nem triviális belépési pont-



1.3. ábra: Egy lehetséges FLOSS belépési pont (szerkesztette a szerző)

tal rendelkezhet, ráadásul ezek a pontok többszörösen visszacsatolásokon keresztül bonyolult függőségi hálózatot alkothatnak. Könnyen előfordulhat, hogy a LIRE biztonsági szintjét egy látszólag független, távoli folyamat befolyásolja. Ilyen lehet például egy másik, a vizsgált LIRE tekintetében kiemelt fontosságúnak tekintett LIRE rendszer információs rendszerében használt dobozos szoftver, amelynek fejlesztése során a gyártó egyik beszállítójának egyedi szoftverében olyan nyílt forrású szoftverforrásokat használnak fel, amelyek függőségei közt egy hibás komponens található (1.3. ábra).

Intuitív módon azt várhatnánk, hogy minél messzebb esik a sérülékenység a végfelhasználás helyétől, annak veszélyessége arányosan csökken, sajnos azonban általában ennek éppen az ellenkezője az igaz. A sérülékenység szempontjából teljesen mindegy, hogyan került a rendszerbe, kizárólag az számít, hogy jelen van és kihasználható. Ugyanakkor a többszörös áttétel jelentősen lelassíthatja a javítások végigfutását a rendszeren, következésképpen a támadáshoz már nem szükséges zero-day[9] sérülékenység, az egyébként már meglévő javítások a rendszer tehetetlensége miatt csak jóval később kerülnek bele a felhasznált szoftverbe.

1.3 Irodalomkutatás

A KC-2 kutatási cél eléréséhez Petersen és társai által javasolt Systematic Mapping Study módszertanát alkalmaztam, amely jól használható, elterjedt módszer a szoftverfejlesztés terén felmerülő osztályozási problémák kezelésére és a téma struktúrálására [19].

A Systematic Literature Review (SLR) módszerével ellentétben a Systematic Mapping Study (SMS) a témákra bontott tudományos bizonyítékok elérhetőségére koncentrál, kevésbé érzékeny a precízen megfogalmazott keresőkérdésekre és nem igényli a publikációk minőségének értékelését [34]. Egy szélesebb terület átfogó elemzésére az SMS megfelelőbb eszköz, ugyanis segítségével hatékony képet alkothatunk az egyes részterületek kutatottsági állapotáról és módszereiről.

A Systematic Mapping Study tervezésének vázlatos lépései az alábbiak:

1. téma és kérdésfelvetés;
2. kutatási terv, források és keresőkifejezés meghatározása;

[9]Bejelentetlen, javítással még nem rendelkező sérülékenység]

3. tanulmánykiválasztási és elutasítási kritériumok meghatározása;
4. adatgyűjtés;
5. adatelemzés;
6. eredmények előállítása.

1.3.1 Kutatási cél

A teljes kutatómunka kérdésfelvetésével összhangban az SMS tanulmány az KC-2.1 kutatási célt hivatott megvalósítani, azaz felméri, hogy a FLOSS egyes hatásai mennyira kutatottak a különféle területeken. Az SMS tanulmány által összegyűjtött anyagok egyúttal a jellegzetességek rendszerezésének (KC-2.2) és a sajátosságok meghatározásának (KC-2.3) alapját is képezik.

A kutatási kérdés további bontásával az alábbi részcélokat határoztam meg:

- KC1.1a: A nyílt forrású szoftverek biztonsággal kapcsolatos egyedi sajátosságainak meghatározása.
- KC1.1b: A kutatói közösség milyen súllyal foglalkozik az egyes sajátosságokkal?
- KC1.1c: Milyen típusú kutatásokat végeztek a témában?
- KC1.1d: Milyen témákkal foglalkoznak a kutatók?
- KC1.1e: A jelenlegi FLOSS biztonsággal kapcsolatos eredmények milyen biztonsági hatásokat azonosítanak?

1.3.2 Alkalmazott protokoll

A kereséseket a következő digitális könyvtárakban végeztem :

- IEEE Digital Library,
- ACM Digital Library,
- ScienceDirect,
- és SpringerLink.

A kiválasztott digitális könyvtárak nagy mennyiségű művet indexelnek és vélhetően jól lefedik a területen folyó mértékadó kutatásokat. Az adatgyűjtésben valamennyi, tehát nem csak a szabadon elérhető (Open

Access) publikációk kerültek bele, ami sajnálatos módon csökkenti a kutatás reprodukálhatóságát, de úgy véltem, a szabadon elérhető művek relatív alacsony száma miatt a területet csak a teljes anyagmennyiséggel lehet megfelelően lefedni. A különböző könyvtárak azonos művekre is hivatkozhatnak, ezért a keresés után duplum-ellenőrzésre volt szükség. Az SMS vizsgálatot 2016 elején végeztem, ennek megfelelően az SMS eredményeiben csak 2016 márciusa előtti művek szerepelnek.

A források meghatározása után a következő feladata a tanulmányok kiválasztási és szűrési kritériumainak definiálása. A SMS kutatás a következő kiválasztási feltételek szerint végeztem:

- digitális könyvtárakban kereshető dokumentumok, cikkek folyóiratok és konferencia publikációk amelyek a nyílt forrás vagy annak fejlesztési módszertanának valamilyen biztonsági vonatkozását elemzik vagy összehasonlítják azt a zárt forrású fejlesztési módszertan eredményeivel.
- 2005 után jelent meg. Ennél régebbi publikációk esetén kicsi az esély, hogy olyan releváns információ található a gyorsan fejlődőt területről, ami később nem jelenik meg sehol.
- A tanulmány lehet kísérlet, esettanulmány, összehasonlító elemzés, vélemény, tapasztalati beszámoló.
- computer science, software engineering és security profilú folyóiratban jelent meg.

A kihagyási kritériumok a következőek voltak:

1. Nem angol vagy magyar nyelven íródott.
2. Hálózaton keresztül nem elérhető. Azaz a teljes anyag nem beszerezhető digitális könyvtárak vagy nyílt oldalon keresztül.
3. Invited papers, keynote speeches, szabványok, workshop reports, reviews.
4. Nem teljes értékű dokumentumok, vázlatok, prezentáció diák és abstractok. Korai elérésű művek.
5. Másodlagos és harmadlagos tanulmányok és meta analízisek.
6. Nem szoftvertervezéssel foglalkozó computer science témájú cikkek (pl. database systems, human-computer interaction, computer networks)
7. Nyílt forrásra csak további, tervezett feladatként hivatkozó munkák.
8. A publikált eredmény nem köthető a nyílt forráshoz, pusztán a bemutatott vagy felhasznált szoftver(ek) nyílt forrású(ak).
9. nyílt forrású programot elemez, de egyértelműen biztonsági vonatkozások nélkül: teljesítmény elem-

zés, funkcionális értékelés.

10. Azért foglalkozik nyílt forrással mert könnyen elérhető, de a vizsgált kritériumnak nincs köze a biztonsághoz és nem vizsgálja a zárt forrástól való eltérést. (Például olyan egyébként nehezen elérhető adatokat vizsgál mint a commit size, loc, stb.)
11. Esettanulmányok, ahol egy adott open source szoftver adott tulajdonságát elemzik, amely nem általánosítható. (pl. USB kezelés Linux alatt)
12. Egy szűkebb open source kategória elemeit hasonlítja össze. (pl. nyílt forrású játékok, ERP, adatbázisok összehasonlító elemzése).

A információk kinyerését három iterációs lépésben végeztem. A keresőszöveg alapján beazonosított művek feltételeknek való megfelelését a cím és az absztrakt segítségével vizsgáltam. Amennyiben ez alapján a megfelelés nem volt egyértelműen eldönthető, a szöveg átfutásával döntöttem. Második lépésként a bevezetés és a konklúziók alapján besoroltam a megfelelő kategóriákba. A harmadik lépés során a művek teljes átolvasásával beazonosítottam a biztonságra vonatkozó javaslatokat és problémafelvetéseket.

Az adatgyűjtés eredményét SQLite adatbázisban rögzítettem, a szűréseket és vizualizációt saját fejlesztésű szoftverrel végeztem el.

1.3.3 Keresési stratégia

A nyílt forrás szemantikai rendszere nem teljesen kiforrott, több rövidítés és szó is jelölheti ugyanazt a fogalmat, míg egyes szerzők azonos kifejezés alatt mást érthetnek. A nyílt fejlesztési modellel kapcsolatos publikációkat Klaas-Jan Stol szerint az alábbi kifejezés segítségével lehet beazonosítani [35]:

”open source software” OR ”libre software” OR
”free software” OR ”OSS” OR ”FOSS” OR ”F/OSS”
OR ”F/OSSD” OR ”FOSSD” OR ”FLOSS” OR ”F/LOSS”
OR ”OSSD”

Ebből a kifejezésből az “OSS” szöveget Klaas-Jan Stol eltávolította, feltételezve, hogy ha ez a szó szerepel, akkor az első kereső kifejezés szintén szerepel. A keresőkifejezés célja a nyílt és zárt modell eltéréseinek felderítése illetve a biztonsággal kapcsolatot jellemzők meghatározása volt, ennek érdekében a felhasznált

keresőszövegnek tartalmaznia vagy a zárt forrásra vonatkozó utalást vagy a a sérülékenységre, biztonságra, problémákra utaló kifejezéseket kell tartalmaznia. A cél elérése érdekében az alábbi keresőkifejezést állítottam össze:

```
(   "open source software" OR "libre software" OR
    "free software" OR "FOSS" OR "F/OSS" OR "F/OSSD" OR
    "FOSSD" OR "FLOSS" OR "F/LOSS" OR "OSSD"
) AND (
    (
        ("closed source" OR traditional OR proprietary) AND
        (comparison OR evaluation OR difference)
    ) OR vulnerability OR (security AND (implication* OR problem* OR weakness* OR issue*))
)
```

Az IEEE Xplore digitális könyvtár nem volt képes kezelni az összetett keresőkifejezést, ezért a szűrést ebben az esetben két részletben, logikai bontással kellett megvalósítani. Ahol a digitális könyvtár tartalmazott téma szerinti szűrési lehetőséget, a keresést kibővítettem bármely Open Source kulcsszóra amennyiben az SWE és Security témakörbe esett. Ezzel a módszerrel vélhetően sikerült lefedni a témába vágó publikációk döntő részét.

A keresés, duplum eltávolítás és az első kiválasztási forduló alapján elfogadott művek száma a 1.1. táblázatban látható módon alakult.

1.1. táblázat: SMS kiválasztási és elutasítási kritériumok eredménye.

Digitalis könyvtár	keresés	első szűrés
Springer Link CS/SWE	3948	285
IEEE Xplore	256+158	74
Science Direct	2829	103
ACM	4613	144
Összesen:	11804	606

1.3.4 Osztályozási séma

A publikációk rendszerezése érdekében hat kvalitatív és egy kvantitatív (év) osztályt vezettem be. Az osztályozás kategóriáit részben korábbi tapasztalatok, részben egy 50 elemű pilot halmaz segítségével határoztam meg. A kategóriák az irodalomkutatás során ennek ellenére bővültek, ezért a bővítés elötti műveket újrakategorizáltam. Az osztályozás mellett szabad szöveges címkézést is alkalmaztam a kereshetőség megkönnyítése végett. Az alkalmazott osztályokat és multiplicitásukat a 1.2. tábla mutatja be. Az "S''-el jelölt osztályok publikációnként csak egy értéket vehetnek fel míg az "M''-el jelöltek többértékűek.

1.2. táblázat: SMS osztályok

kód	Osztály	típus
Year	publikálási év (Year)	S
TYP	kutatás típusa	S
OSP	előző kutatásban meghatározott FLOSS jellemzők	M
SEC	biztonsági terület	M
CTR	a publikáció eredményének típusa	M
MTH	módszertani kategória	M
TOP	a publikáció témája	S

A kutatás típusa szerinti besorolást Petersen [18] által javasolt felosztás szerint végeztem, a 1.3. táblázatban bemutatott definíciók alapján.

1.3. táblázat: Tudományos művek típus szerinti felosztása

Kód	Típus kategória	Leírás
VAL	Validation Research	Új módszereket vizsgál, amelyeket még nem ültettek át a gyakorlatba. A felhasznált módszer lehet például kísérlet.

Kód	Típus kategória	Leírás
EVA	Evaluation Research	A módszert gyakorlatba is átültették és a értékelték. Bemutatja hogyan működik a módszer a gyakorlatban, és mi az implementáció következményei (előnyök és hátrányok). Az iparban felmerülő problémák azonosítása is ide tartozik.
PRB	Problem Investigation Isme és jellemzőinek	rt vagy újonnan felmerült probléma körülményeinek vizsgálata.
SOL	Solution Proposal	Egy problémára javasol megoldást, amely lehet új vagy egy meglévő jelentős bővítése. Az alkalmazhatóság potenciális hasznát rövid példával vagy jó érveléssel támasztja alá.
PHI	Filozófiai publikáció	Ez a fajta munka új nézőpontból vizsgálja a már létező dolgokat a témát valamilyen taxonomia vagy koncepcionális keretrendszerbe rendszerezve.
OPI	Vélemény	Ezek a munkák valakinek a személyes véleményét fejezik ki, miszerint egy adott módszer jó vagy rossz, illetve mi a helyes eljárás egy adott esetben. Nem függenek más munkáktól és kutatási módszertantól
EXP	Tapasztalat	Ismerteti, mit és milyen módon hajtottak végre valamit a gyakorlatban. A szerző személyes tapasztalatának kell lennie.

A további osztályok értékei a következők:

kód	FLOSS kategória (OSP)
COM	Közösség
DEV	Fejlesztés
DIS	Terjesztés
ECO	Gazdasági hatás
MET	Metaadatok
OTH	Egyéb
POL	Szabályozás
PRD	Termék
SOC	Tudati dimenzió

kód	FLOSS kategória (OSP)
USG	Felhasználás

A biztonsági területek kategóriáit az amerikai MITRE szervezet által használt felosztást követő módon határoztam meg.

kód	Biztonsági terület (SEC)
SWA	Szoftverminőség vizsgálat
SCM	Beszállítói lánc
VUL	Sérülékenységek
RIM	Kockázatelemzés
CFM	Konfiguráció-menedzsment
LCS	Életciklus menedzsment
TST	Tesztelés

kód	Tudományos eredmény kategória (CTR)
PRT	Eszköz: a publikáció felhasználható eszközt ismertet.
PRS	Folyamat: a publikáció folyamatleírást tartalmaz.
MTR	Metrikák: a publikáció empirikus eredményeket közöl numerikus formában.
MOD	Modell: új modellt határoz meg.
MET	Módszer: megoldási módszert javasol egy problémára.
TXN	Taxonómia: új osztályozást javasol vagy taxonomiát fogalmaz meg.

kód	Módszertani kategória (MTH)
CST	Esettanulmány
DAN	Adatelemzés
SRV	Felmérés

kód	Módszertani kategória (MTH)
GRT	Grounded theory
FST	Field study
EXP	Kísérlet
SMR	Systematic review
THW	Elméleti munka
MOA	Modellanalízis
SMS	Systematic Mapping study

kód	Téma kategória (TOP)
ADP	Elfogadottság
CVT	Áttértés zártról nyílt modellre
CPL	Jogi és szabályozási megfelelőség
DEV	Fejlesztés
ECO	Gazdaságosság
OIT	FLOSS integráció
RSR	FLOSS kutatás
OTH	Egyéb
SEC	Biztonság
EVO	Szoftver életút, fejlődés
QAA	Minőségvizsgálat
GEN	Általános OSS tulajdonságok

1.3.5 Eredmények

A publikációk osztályozását egy erre a célra fejlesztett web alapú űrlapon keresztül végeztem, a nyers eredményeket további feldolgozásra SQLite adatbázisban tároltam. Az iteratív címkézési eljárás eredményeképpen létrejövő adatbázisból SQL lekérdezésekkel állítottam elő a végső eredményeket. Az numerikus eredményeket az értekezésben halmozott oszlop és buborék diagramok segítségével prezentáltam. Az ere-

deti nyers adathalmaz valamint a kiértékeléshez és az ábrák készítéséhez használt szoftver forráskódja kutatás szoftvertárházán keresztül szintén elérhetőek.

Az egyes FLOSS sajátosságok kategóriáiba eső publikációk számának évenkénti bontását a 1.4c ábra mutatja be (KC1.1a, KC1.1b). Kutatottság szempontjából legjelentősebb sajátosságok a közösség szerepe és a fejlesztési módszertan eltérései. A fejlesztési eltérésekkel foglalkozó publikációk 2009 környékén tetőztek, de a kérdés vizsgálata folyamatosan jelentős részét képezi az összes kutatásnak. Láthatóan minden évben állandó szereplők még a FLOSS sajátos felhasználási problémáit boncolgató munkák. Az egyes kategóriákba eső publikációk számát tekintve nincs jellemző tendencia, tehát nem állíthatjuk, hogy egyes sajátosság az előtérbe kerültek vagy háttérbe szorultak volna.

A publikáció típusa szerinti évenkénti bontást a 1.4a ábra mutatja be. Jól látható, hogy 2008-tól kezdve az uralkodó kutatási módszer a kiértékelő kutatás (Evaluation Research). A tapasztalati eredmények közlése közel változatlan és a publikációk mintegy ötödét teszi ki. Jelentős szerepet játszik a közvetlen javaslatokat ismertető Solution Proposal kategória, amely szintén közelítőleg azonos részarány jellemez a vizsgált időintervallumban. Mint arra számítani lehetett a tisztán elméleti munkák száma elenyésző és csak néhány évben jelenik meg. A kutatói közösség a vizsgált intervallumban elsősorban a probléma megértésére koncentrál.

A publikációk téma alapú bontását tekintve – melyet a 1.4b. ábra mutat be – látható, hogy 2008-tól kezdve az uralkodó kutatási terület a nyílt forrású fejlesztés (DEV), amely az összes publikáció mintegy 40-50%-át adja. A téma magas aránya nem véletlen, hiszen a nyílt forrás legfőbb egyedi jellegzetessége minden valószínűség szerint fejlesztési módszertanából következik. Az alkalmazással és elfogadottsággal kapcsolatos cikkek (ADP) száma 2011 körül tetőzött, ezt követően visszaesett, feltehetően a nyílt forrás fogalmának széles körben ismertté válása következtében. A minőségvizsgálattal kapcsolatos cikkek (QAA) folyamatosan szerepelnek a tudományos palettán, ami arra enged következtetni, hogy az ipar és a tudományos közösség érdeklődése élénk, ugyanakkor sok kérdés egyelőre nem tisztázott a területen. A kifejezetten biztonsági fókuszú publikációk (SEC) száma erősen ingadozó képet mutat, igaz, a téma folyamatosan jelen van. Ez arra enged következtetni, hogy a biztonság kérdése háttérbe szorul a nyílt forrás egyéb hatásainak vizsgálata mellett. Bár a gazdaságosság és a nyílt forrásra való áttérés (vagy éppen visszatérés) kérdése igen gyakran foglalkoztatja a hírportálok cikkíróit, a tudományos közösség ezekkel a témákkal csak marginálisan

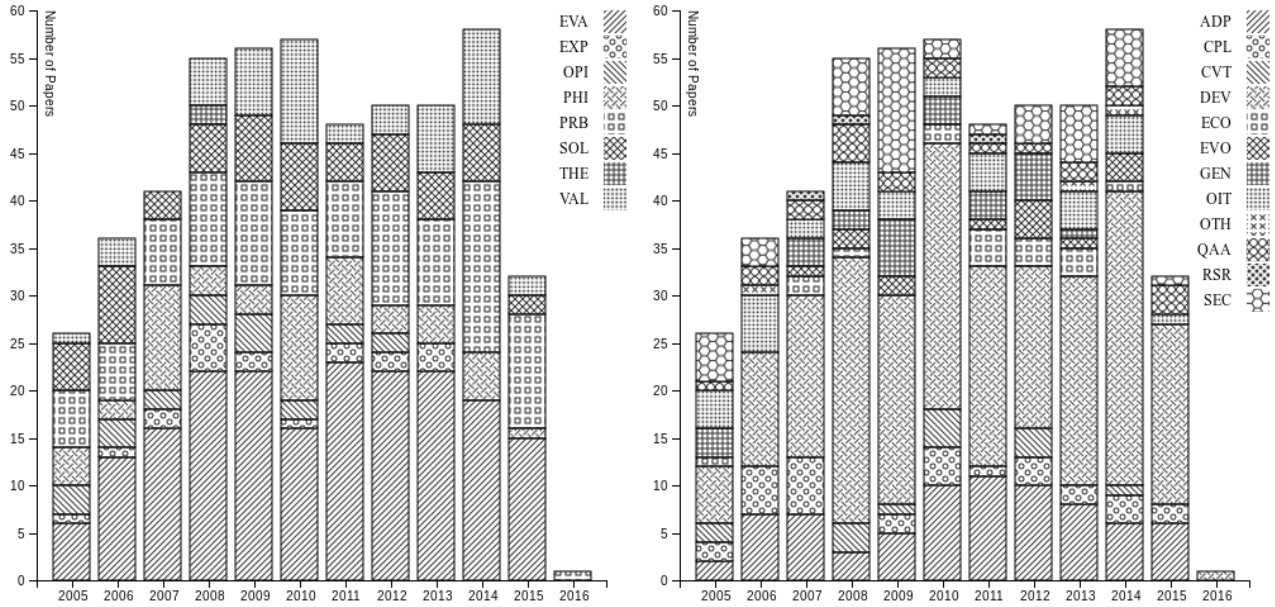
foglalkozik.

A 1.4d. ábra az alkalmazott módszereket mutatja be évenkénti bontásban. Az uralkodó alkalmazott módszer az esettanulmány, az adatelemzés és a felmérés. Az esettanulmányok száma csökkenő tendenciát mutat, helyét lassan átveszi az adatelemzés. A felmérések továbbra is jelentős szerepet töltenek be a kutatásokban, ami kiemeli a nyílt forrás tudati dimenziójának jelentőségét. A kérdőívek viszonylag magas száma nem meglepő. Számos jellegzetesség nehezen vagy egyáltalán nem mérhető, esetleg egyenesen szubjektív, így vizsgálata inkább tartozik a társadalom-tudomány mint a mérnöki tudományok hatókörébe. Tekintve azonban, hogy a biztonság definíció szerint hasonlóképpen szubjektív jellegű,

Érdekes kérdés, hogy az egyes részterületeken mely kutatási módszerek az irányadóak. A 1.5a. ábra bemutatja, hogy mely FLOSS kategóriában hány publikáció alkalmazott egy bizonyos kutatási módszert. Mint az korábban kiderült, az uralkodó kutatási módszerek az esettanulmány a felmérés és az adatelemzés, de az eloszlásuk nem teljesen azonos a FLOSS minden területén. A tudati dimenzió és felhasználási kérdéseket elsősorban kérdőíves módszerrel igyekeznek meghatározni – mint az várható volt – ezen a területen az esettanulmányok és adatelemzések száma alacsonyabb, minden más módszer elenyésző. Elméleti munkát kizárólag a fejlesztés és a módszertan területein találunk, más kategóriákban nincsenek formális módszereket alkalmazó vagy javasló megoldások. A metaadatok területén – tekintettel a nagy mennyiségű, könnyen feldolgozható adatra – az uralkodó módszer az adatelemzés, de némiképp meglepő módon a kérdőíves módszereket itt is előszeretettel alkalmazzák.

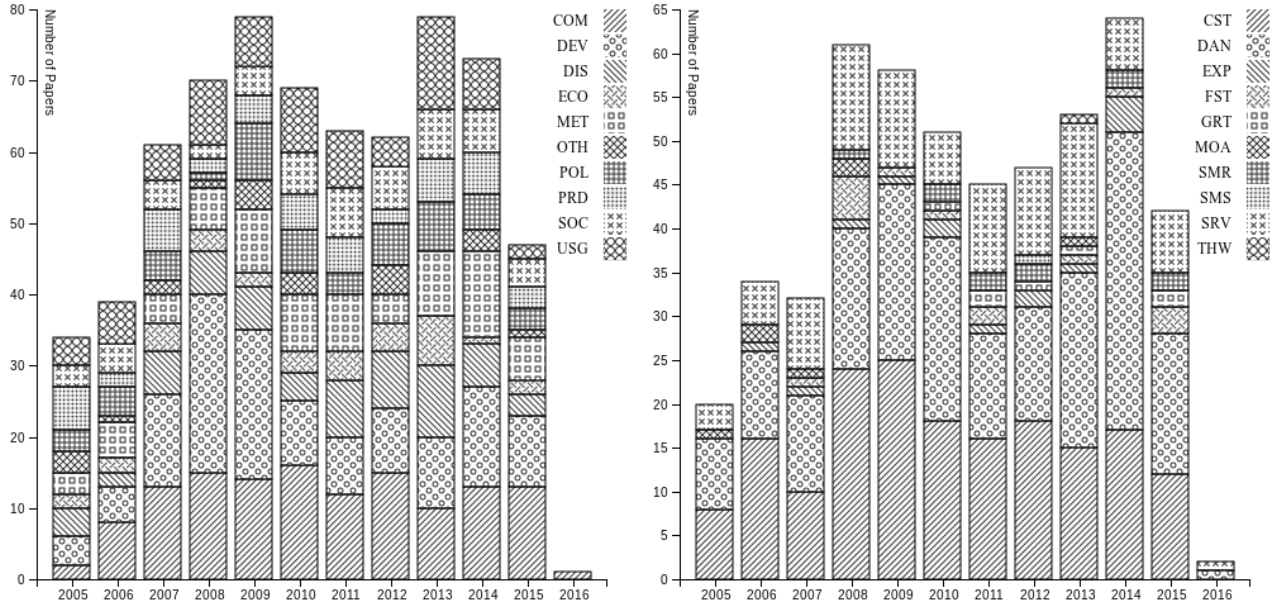
A kutatás szempontjából legnagyobb fontossággal bíró terület a biztonság, amelynek módszereit külön vizsgáltam (1.5b. ábra). Az alkalmazott módszerek eloszlását tekintve itt sem tapasztalható jelentős eltérés, esettanulmányok, adatelemzések és felmérések adják a publikációk nagy részét. Érdekes megfigyelni, hogy bizonyos módszerek, mint például a SMS/SMR, modellanalízis, elméleti munkák szinte teljesen hiányoznak ezen a területen. Közvetlen tapasztalatszerzésből (Field Study) származó eredményeket csak a szoftverminőség-vizsgálat, beszállítói lánc elemzése és a sérülékenység-elemzés területén találunk, igaz ezeken a területeken a minták száma túl alacsony ahhoz, hogy az eredményt reprezentatívnak tekinthessük. Összességében elmondható, hogy a FLOSS biztonsággal kapcsolatos kutatások döntő része (jóval több mint fele) Szoftverminőség-vizsgálattal kapcsolatos esettanulmány és adatelemzés.

Fontos kérdés, hogy az egyes részterületeken milyen típusú eredményekkel találkozunk. A 1.5c. ábra az



(a) Publikáció típusa szerint

(b) Publikáció témája szerint



(c) FLOSS tulajdonság szerinti osztályozás

(d) Módszertan szerinti osztályozás

1.4. ábra: SMS osztályok kategóriái évenkénti bontásban (szerkesztette a szerző).

eredmények típusait mutatja be FLOSS részterületek szerinti bontásban. Mint látható a tanulmányok eredményeképpen első sorban metrikákat, modelleket és módszereket kapunk, a folyamatleírások, taxonómiák és eszközök ismertetésének száma elenyésző. Numerikus adatokat döntő részben a közösség, fejlesztés és metaadatok kategóriában találhatunk, ami érthető, hiszen a nyílt forrás jellegzetességének számító nyílt fejlesztési modellnek hála ezek a numerikus adatok könnyen hozzáférhetők. A javasolt módszerek száma meglepő módon egyenletes, szinte minden kategóriában hasonló, azaz annak ellenére, hogy a forrásadatok elérhetősége és a módszertan eloszlása kimutathatóan kiegyensúlyozatlan, a kutatók igyekeznek minden részterületen módszereket javasolni. Mindez arra enged következtetni, hogy a részterületek fontosságát a kutatók ismerik, de megfelelő minőségű háttéradatok híján jóval kevesebb megalapozott tudományos eredményt lehet ezeken a területeken felmutatni.

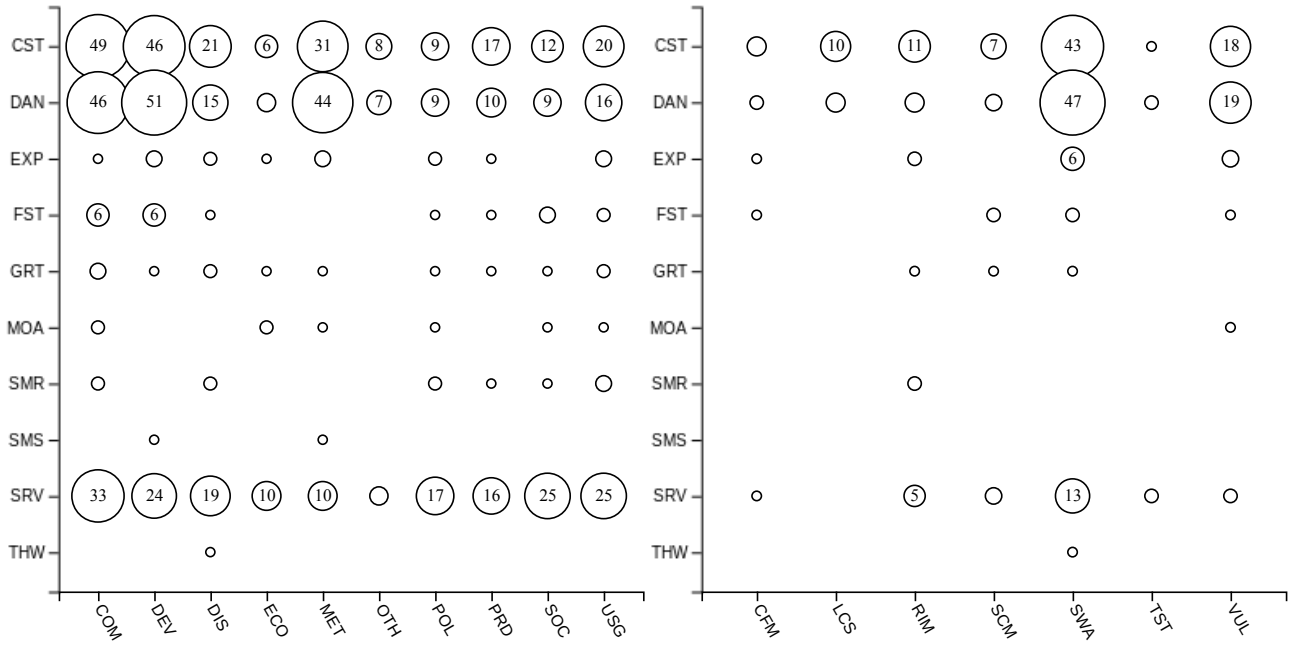
A 1.5d. ábrán látható milyen típusú kutatások célozzák az egyes biztonsági területeket. Szembeötlő, hogy a biztonság témáját feldolgozó publikációk jelentős részét a szoftverminőség vizsgálat területén már alkalmazott megoldások értékelése (Evaluation Research) adja. Viszonylag jelentős az ugyanezen területen végzett probléma és validációs vizsgálatok, megoldási javaslatok száma, illetve a sérülékenységek területén alkalmazott gyakorlati módszerek vizsgálata. Ugyanakkor feltűnő, hogy sok a “fehér folt”, azaz bizonyos biztonsági területeken egyes módszerek teljesen hiányoznak. Nagyon kevés publikáció foglalkozik a konfiguráció menedzsment és a tesztelés területével. A kockázatelemzés területén, más kategóriáktól eltérően, a problémák vizsgálata és a javasolt megoldások az uralkodók.

1.3.6 Összefoglaló analízis

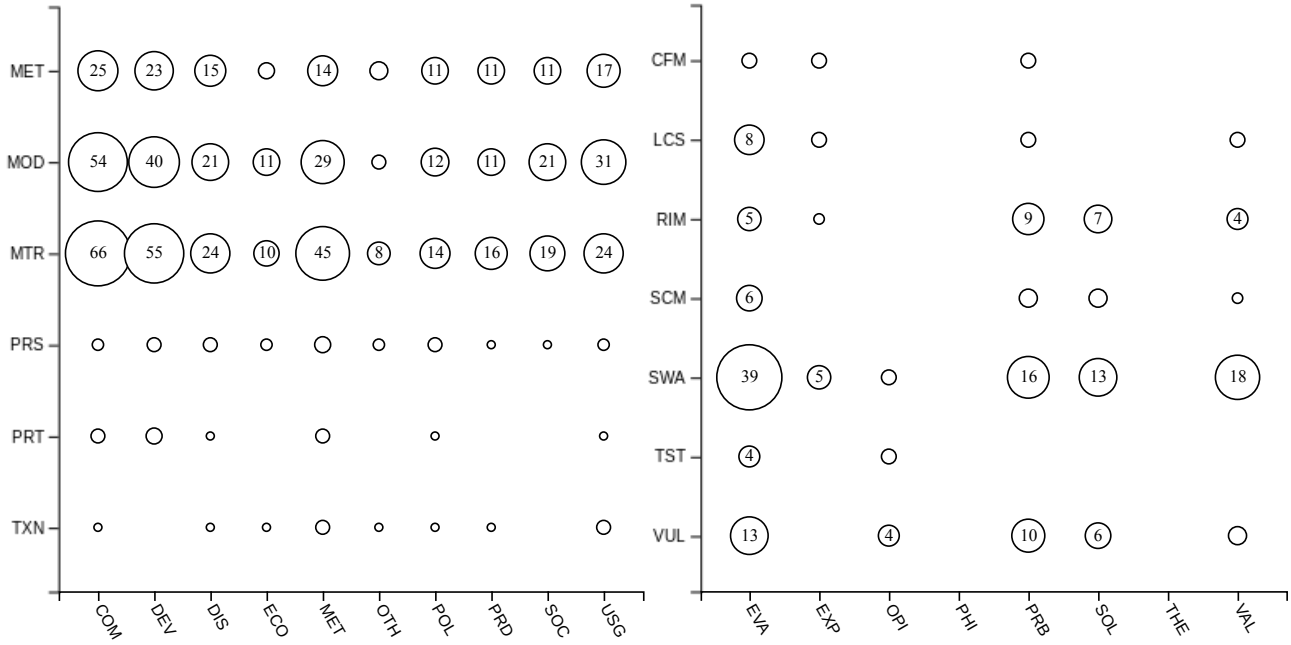
Megállapítottam, hogy a leginkább kutatott FLOSS sajátossági kategória a fejlesztési módszertan különbsége. Természetesen ebből nem következik, hogy a biztonságra ez a sajátosság gyakorolja a legnagyobb hatást, de – legalábbis a publikációk száma alapján – ez a kérdés a legjobban kutatott, itt várható stabil eredmény, a lehetséges biztonsági problémák legnagyobb lefedettsége.

A vizsgált időszakban a tudományos közösség a probléma megismerésére, megértésére koncentrál. Erre utal a tájékozódó módszerek (esettanulmány, felmérés) dominanciája és az uralkodó publikáció típusok (tapasztalatok, értékelések, feltérképezés) száma szemben az elméleti munkák alacsony részarányával.

Úgy találtam, hogy a javasolt módszerek száma egyenletes, bár a forrásadatok elérhetősége és a módszertan



(a) FLOSS sajátosság szerint alkalmazott módszertan (b) Biztonsági területenként alkalmazott módszertan



(c) Publikáció típusonként jellemző eredményfajták (d) Publikáció típusok biztonsági területenként

1.5. ábra: Publikációk száma az osztályok kategóriái szerint (szerkesztette a szerző).

eloszlása kimutathatóan kiegyensúlyozatlan. A kutatói közösség valamennyi FLOSS területet fontosnak tartja, megoldási javaslatokkal is él, ám megfelelő alapkutatások híján bizonyos területeken igen nehéz tudományos eredményeket felmutatni.

A FLOSS biztonsági kérdéseivel kapcsolatos kutatások többnyire szoftverminőség-vizsgálattal kapcsolatos esettanulmányok és adatelemzések. A tudományos közösség a szoftverminőség-vizsgálatra és sérülékenységekre koncentrál, egyéb biztonsági területek (pl. tesztelés, életciklus menedzsment, konfiguráció menedzsment stb.) meglehetősen alacsony. Megállapítom, hogy a FLOSS biztonsági kérdéseinek kutatása a vizsgált időszakban egyoldalú és néhány részterületre fókuszált.

A szűrés és elemzés során számos olyan új sajátosság is felmerült, amely a kutatás korábbi szakaszából származó rendszerezésbe nem volt beilleszthető. Ezeket a jellemzőket a taxonómia bővítésével kezeltem, illetve felvételre kerültek a biztonsági hatásokat nyilvántartó adatbázisba. Ilyen, korábban rendszerzetlen jelentősebb sajátosság volt például a preprocesszor kapcsolók használatának lehetősége, a fejlesztői függetlenség és az ezáltal kialakuló fejlesztői befolyás kérdése, a Trusted Platform technológiákkal való esetleges inkompatibilitás kérdése illetve az egyszerű hozzáférhetőség folytán kialakuló kódmásolás (kódbeszivárgás) problémaköre. A felderített biztonsági hatásokkal részletesen a 2. fejezetben foglalkozom.

1.4 Részkövetkeztetések

A KC-1 célkitűzés keretében a LIRE és a FLOSS elemek kapcsolatát vizsgáltam. Megállapítom, hogy a LIRE többszörös függősége és a FLOSS bujtatott jelenléte folytán a biztonságra gyakorolt hatás rendkívül összetett, elméletileg tetszőleges számú közvetítőn keresztül is létrejöhet. A feltörekvőben lévő felhő és IoT technológiák terjedésével a helyzet különösen kiéleződhet, tekintettel e területek jelentős (gyakran rejtett) FLOSS felhasználására.

A LIRE direkt FLOSS kitettsége az alábbi négy szint valamelyikén valósulhat meg:

- **I. típus**: A szervezet ületi termékként, viszonteladótól szerzi be a nyílt forrású szoftvert avagy hardver komponensként, nagyobb szoftvercsomag részeként illetve önálló megoldásként alkalmazza azt.
- **II. típus**: A szervezet közvetlenül a fejlesztőközösségtől szerzi be a szoftver terjesztésre szánt változatát, csomagkezelő rendszeren keresztül vagy direkt módon, általában bináris formában.

- **III. típus**: A szervezet saját belső előírásai szerint fordítja le a FLOSS termék stabil vagy fejlesztői változatát az eredeti forráskód felhasználásával. A szervezet ez esetben képes a fordítási opciók illetve akár a forrás módosítására.
- **IV. típus**: A szervezet direkt vagy közvetett módon részt vesz a fejlesztésben, saját fejlesztési ágat vezet, egyedi módosításait a szülő projektbe visszavezeti vagy folyamatosan karbantartja.

Az áttételes hatás tényleges megvalósulási gyakoriságának vizsgálata sajnos objektív akadályokba ütközik, tekintettel a LIRE pontos felépítésének érzékeny és minősített voltára. Hazai LIRE rendszerek üzemeltetőivel folytatott személyes találkozók keretében meggyőződhettem róla, hogy a kutatásban feltárt hatáspontok legalább egy része mindenképpen valós lehetőség, továbbá a 2. fejezetben bemutatok néhány konkrét példát is.

A KC-2.1 célkitűzésnek megfelelően megállapítottam, hogy a tudományos közösség jelentős túlsúllyal foglalkozik a FLOSS fejlesztési módszertan kérdéseivel, míg más aspektusok (biztonság, megfelelőség, életciklus menedzsment) kevésbé kutatottak. A biztonsági kérdéseket is érintő munkák döntő részben a nyílt forrású termékek szoftverminőség-vizsgálatára szorítkoznak. A könnyen hozzáférhető forráskód okán gyakran szerepelnek kutatások adatforrásaként, de a nyiltság konkrét hatásainak vizsgálatával a publikációk általában nem foglalkoznak.

A javasolt módszerek egyenletes eloszlást mutatnak a sajátosságok tekintetében, azaz a kutatói közösség, valamennyit fontosnak tartja, ugyanakkor empirikus adatok és alapkutatási eredmények első sorban a nyílt forrású fejlesztés területén állnak rendelkezésre, ezért az eredmények döntő része is ide koncentrálódik.

Megállapítottam, hogy a gazdasági hatással, tudati dimenzióval, terméktulajdonságokkal kapcsolatos jellemzők vizsgálatához szükséges adatot jelenleg kizárólag szük területre koncentrálódó (nem reprezentatív) felmérésekből lehet gyűjteni vagy egyáltalán nem áll empirikus eredmény rendelkezésre és csak szakértői véleményekre támaszkodhatunk.

A kutatás első fázisában többszöri finomítással létrehoztam a FLOSS sajátosságok egy lehetséges rendszertanát (0.5 ábra) amely megfelel a KC-2.2 célkitűzésnek. Ezt a rendszertant alkalmazom az következő fejezetekben.

Megállapítom, hogy a FLOSS mint jelenség LIRE-re gyakorolt hatásainak vizsgálatához semmiképpen

sem elegendő a LIRE elsődleges rendszereinek vizsgálata. A FLOSS esetleges sérülékenységei akkor is jelentős hatást gyakorolhatnak a LIRE komponenseire, ha az információs rendszer közvetlenül (papíron) semmilyen FLOSS megoldást nem használ.

2 FLOSS Sajátosságok

2.1 Fejlesztési folyamatok (FS-F)

Az üzleti szoftverfejlesztési folyamatok nem igazán illeszkednek a klasszikus nyílt fejlesztési közösségekre [36–38], a két modell ugyanis kulturálisan különbözik. Míg az zárt modell a fejlesztők ellenőrzésén, fix ütemezése és minőségbiztosítási elveken alapul[39], a nyílt modell a szabadságot, az önszerveződést és az innovációt helyezi előtérbe. Napjainkban a helyzet persze már nem ilyen egyértelmű, hiszen a klasszikus nyílt modell mellett egyre gyakrabban találkozunk cégirányítás alatt álló vagy szponzorált FLOSS projektekkel, amely a delegált fizetett fejlesztőkön keresztül a hagyományos szoftverfejlesztés módszertanát vegyíti a közösségi modellel [29, 40]. Sőt, ez a folyamat kétirányú, az üzleti fejlesztésben is terjednek a nyitottabb módszerek (SCRUM, Kanban) valamint egyre népszerűbb a nyílt modell elveit használó Inner Source irányzat[41, 42] is. Az alábbiakban elsősorban a klasszikus önkéntes nyílt modell jellemzőre koncentrálok, hiszen innen ered a FLOSS modell eltéréseinek döntő része.

A nyílt modellt gyakran támadják amiatt, hogy a hagyományos szoftverfejlesztés számos fontosnak tartott eleme, mint a formális követelmény-specifikáció, architektúra dokumentáció vagy minőségbiztosítási szabályozás messze elmarad az üzleti világban elvárt szinttől illetve sok esetben teljesen hiányzik. A hibrid vagy szponzorált nyílt fejlesztési közösségek üzleti szemlélete segít kiküszöbölni a hiányosságok egy részét[43], ehhez azonban a szervezeteknek megfelelő befolyást kell szereznie a projekt felett. Csakhogy a túlzott befolyás nem előnyös, elriasztja a fejlesztők egy részét [44] illetve extrém esetben forkot kényszeríthet ki (pl. MariaDB vs. Oracle MySQL). A fork egyik gyakori célja a projektmenedzsment megváltoztatása [45].

Valójában a tradicionális nyílt közösség önmagában is képes a zárt modellel azonos minőségű kódot elő-

állítani, az eredmény és a növekedés üteme általában nem lesz látványosan jobb, de rosszabb sem [46, 47]. Ez a teljesítmény azonban egyáltalán nem a véletlennek vagy a zseniális elméknek, hanem egy eltérő, de éppoly jól szervezett és fegyelmezett módszertannak [48] köszönhető amely néhány tekintetben hasonlít a zárt modellből ismert Agilis Szoftverfejlesztés[1] módszertanához ám nem azonos azzal[49].

A szoftverfejlesztés hagyományos pillérei a követelmény-specifikálás, tervezési szakasz, a programfejlesztés, tesztelés, kiadás, karbantartás és folyamatot összefogó minőségellenőrzés. A nyílt modell ezeket gyors kiadási ciklusok alkalmazása mellett egymással párhuzamosan végzi, az egyes folyamatok nem is különülnek el élesen.

2.1.1 Követelmények meghatározásának eltérései (FS-F-K)

Több publikációban is megjelenik a FLOSS és üzleti projektek követelmény-rendszere közötti eltérés kérdése. Egyértelmű különbség mutatkozik a projektek magasabb rendű céljai, e célok megfogalmazásának mikéntje, a célok felhasználó-centrikussága és a célok elérésének határideje terén. Az alábbiakban ezeket a tényezőket mutatom be.

Funkcionális követelmény meghatározás eltérései. Üzleti környezetben általában megengedhetetlen a célok hanyag megfogalmazása, míg egy FLOSS projekt megelégedhet vázlatos funkcionális követelményspecifikációval [50, 51] ami csak idővel tisztul le [52] megfelelő mértékben. Más részről nyílt modell gyakran emlegetett jellegzetessége miatt – nevezetesen, hogy a fejlesztők egyben felhasználók is – a funkcionalitás stratégiai céljai már a kezdetektől sokkal tisztábbak, nem a fejlesztő és a vásárló hosszú és nehézkes egyeztetése során jönnek létre [53]. A FLOSS fejlesztésben tehát az informális célok általában világosak, de nem jellemző viszont a mérhető célok definiálása [54] a követelmények kidolgozása során. A klasszikus FLOSS projekteben általában nincsen formális követelményrendszer, a fejlesztési irányokat közösségi vita dönti el [55–57], ezzel ellentétben az üzleti irányítású közösségekben (pl. Android) a célok többnyire világosak, meghatározásuk semmiben nem tér el az üzleti változatokétól, ami nem véletlen, hiszen itt az irányítás egy piaci szereplő kezében van [58].

Időbeli célok eltérései. A FLOSS projekteket nem motiválja az időben történő piacra kerülés. Ez egyrészt

[1] Angol nevén Agile Software Development, szoftverfejlesztési módszerek egy csoportja, ahol a megoldásokat együttműködés révén önszerveződő multifunkcionális csoportok valósítják meg.

biztonsági szempontból kedvező, hiszen nincs határidő nyomás a fejlesztőkön, felhasználói részről viszont kedvezőtlen, hiszen esetleg hosszú időn keresztül nem jutunk hozzá a kívánt funkcióhoz[59].

Felhasználó centrikusság. A FLOSS fejlesztés alapvetően nem felhasználó-centrikus. Amennyiben kevésbé technikai felhasználók (azaz a fejlesztők halmazán kívül esők) igényelnek valamilyen funkciót, az vagy bekerül a specifikációba vagy nem, szabad kapacitástól függően, ami jelentős különbség ez az üzleti világhoz képest, ahol a felhasználói igény kielégítése a fejlesztés elsődleges iránya. A nyílt modellben nem cél a vásárló elégedettsége [60, 61], az igényfelmérést és követelmények meghatározását programozók és nem erre szakosodott szakemberek végzik [54] és általában hiányzik az igényekre vonatkozó komplex adatgyűjtés [62] is. A felhasználók brainstorming jelleggel, levelező listákon és fórumokon keresztül adhatnak hangot igényeiknek, bizonyos mértékig részt vállalva a projekt tervezésében (legalábbis nagyobb mértékben mint a zárt modell esetén [63]).

Ezt a tervezési módszertant számtalan zsákutca, elvetélt változat és igen lassú haladás jellemzi [64, 65] így hatékonysága és eredményessége megkérdőjelezhető [66, 67], ugyanakkor igaz, hogy ezáltal egy sokkal nagyobb halmaz – a felhasználói bázis jelentős része – vesz részt az innovációban [68]. A funkcionális igény lassú teljesülése LRE rendszerek esetén problematikus lehet, különösen ha ez a hiányzó igény pont a biztonság. A felhasználók alacsony érdekérvényesítő képességére vezethető vissza a FLOSS használhatóságbeli problémái [69] hiszen a technikailag képzetlenebb felhasználó és a fejlesztést irányító fejlesztő-felhasználó számára a jó használhatóság merőben mást jelenthet (felhasználói interfész kidolgozottsága, beállítási lehetőségek egyszerűsége, beépített segítség stb. vonatkozásában).

Többnyire az üzleti termékek felhasználóinak sincs lehetőségük beleszólni a fejlesztésbe, de általában bíznak benne, hogy a piaci végül a megfelelő irányba tereli a fejlesztést[70]. FLOSS esetén az lesz az irány, amit a "látnok" kijelöl, vagy amit a többség megszavaz (lásd FS-K-Sz). Emiatt az üzleti felhasználóknak nagyon fontos, hogy megfelelő lobbierővel rendelkezzenek, amit támogatással, de leginkább a közösség munkájában való részvétellel érthetnek el. Nem véletlen, hogy az igazán nagy piaci értéket képviselő FLOSS projektekben (Linux, cloud technológiák területe) általában szerepet vállalnak az technológiát felhasználó nagyvállalatok illetve – többnyire konzorcium formájában – egész egyszerűen átveszik annak vezetését.

Biztonsági szempontból a felhasználó-centrikusság és a piacorientáltság hiánya nem feltétlen hátrány. A felhasználók igazoltan hajlamosabbak fizetni az érdekes funkciókért mint a biztonságért. Egy biztonsági

hiányosságokkal terhelt de funkciógazdag termék jobban eladható, mint egy biztonságos, ám funkciószegény változat, még akkor is, ha a hiányzó funkciókat a felhasználók valójában nem is használják [71].

Hosszú távú tervezés . A FLOSS közösség alapvető stratégiai célja nem a profit, az időben való piacra kerülés vagy egy kitűzött specifikáció megvalósítása, hanem az együttműködés egy ötlet körül [72–74]. A piaci céloktól való függetlenség pozitív hozadéka az jobb hardver kompatibilitás. Egy-egy hardver támogatottsága nem tűnik el pusztán azért, mert már nem gazdaságos azt fenntartani[75]. Egy Linux rendszert mind a mai napig le lehet fordítani i386-os architektúrán, Gravis Ultrasound támogatással, holott ezt a hardvert lassan már csak múzeumokban találjuk meg.

A fejlesztés hangsúlya általában az innováción és nem a stabilitáson van [76, 77], bár ezt a problémát a legtöbb projekt stabil, teszt és a legfrissebb "bleeding edge" verziók használatával igyekszik enyhíteni.

A célok és követelményrendszer megfogalmazásának fent említett eltérése a FLOSS idealista fejlesztőközössége és a cégek vezetőinek gondolkodási különbségeire vezethető vissza. A cégvezetők gyakran nem kevés gyanakvással tekintenek a FLOSS projektekre az üzleti gondolkodástól elütő idealizmusuk miatt [78].

A követelményrendszer különbözőségének kezelésére esetleg megoldást jelenthetnének a már ma is létező szabadúszó licit[2] vagy crowdsourcing rendszerekhez hasonló platformok kialakítása, ahol a fejlesztésben részt nem vállaló felhasználók a hiányzó funkciókra vagy magára a projektre licitálhatnának. Ez egyúttal a FLOSS fejlesztések finanszírozási problémáin is segítene. Az ilyen irányú próbálkozások a FLOSS esetében egyelőre gyerekcipőben járnak (élő példa többek közt az IssueHunt), egy jól működő modellhez a technikai lehetőségen túl valószínűleg mélyreható társadalmi, hozzáállásbeli változások szükségesek.

Jelenleg a projekt céljainak befolyásolásra az egyetlen gyakorlatban is kivitelezhető megoldás úgy tűnik a projektben való részvétel [79].

2.1.2 Tervezés (FS-F-P)

Általános nézet, hogy a FLOSS projektek nem alkalmaznak formális tervezési módszereket, sőt egyes esetekben a tervezők egyenesen másodrendű állampolgárok a közösségben[80]. A projektek gyakran azonnal

[2] Az un. freelancing vagy microjob platformokon különféle részfaladatokra jutalmat kitűzve szabadúszókat bérelhetünk. Ilyen platform a Freelancer.com, Truelancer, Upwork, Guru, Zeerk.

programozással indulnak, átfogó tervek csak a fejlesztés folyamán vagy egyáltalán nem születnek[54]. A modellek, különösen a dokumentált modellek használata ritka [81], ami problémát jelenthet ha a projektet átlátó régi fejlesztők helyére újaknak kell lépniük, hiszen a betanulási idő igen hosszúra nyúlhat [82].

A közösség alapvetően projekt és nem szervezet központú[83] a fejlesztők javításokról vagy funkciókról blogolnak, menedzsment kérdésekről csak egészen elvétve esik szó [84]. Jobbára nincs definiált módszertan [37, 51, 54], a fejlesztési stratégia dokumentációja többnyire kimerül valamilyen programozási stílus útmutatóban (ami viszont igen gyakori) [85]. Ha van is formális követelményrendszer, kisebb projektekben ritkán tartják be és még ritkábban ellenőrzik [86].

Az önállóan választott feladatok, a szerepkörök egybemosódása kockázatot hordozhat. Nem érthet mindenki mindenhez. A kivitelező szakemberek nem feltétlen képesek jó modelleket alkotni, ahogy a modellező szakember sem biztos, hogy képes jól végrehajtani azt [87]. A FLOSS közösségek ritkán fogalmaznak meg formális biztonsági követelményeket, a meglévő biztonsági minősítő módszerek is stabil tervezési és fejlesztési folyamatokat feltételeznek, ami megnehezíti a FLOSS biztonsági tanúsítását [55]. Minthogy a megszokott módszerek nehezen használhatók a tanúsítást a kód és a metaadatok elemzésével lehet elvégezni [88].

Az tervezésre jellemző a nagy fokú modularitás és a komponensek extenzív használata [36, 89] amit az adapter pattern gyakori jelenléte is jelez [91] ugyanakkor jelentős komplexitást vezet be és verzió-frissítési problémákhoz vezethet. A FLOSS általában valóban modulárisabb mint a zárt modell [60] amire szükség is van, hiszen a nagy számú fejlesztő csak így tud hatékonyan együtt dolgozni elfogadhatatlanul magas integrációs és koordinációs költség nélkül[92, 93]. Az újonnan érkezők az őket érdeklő részre tudnak fokuszálni, anélkül hogy a többiek munkáját zavarnák [37]. A modularitás további előnye, hogy egyetlen jól meghatározott feladatot végző könyvtárban könnyebb megtalálni a hibát, mint egy komplex rendszerben[94]. A modularitás mérhető [95], a mérőszámok segítségével következtetéseket vonhatunk le a projekt sikerességére vonatkozóan.

A kódbázisban erős kód-coupling[3] figyelhető meg, ami részben a központi tervezés és modellező eszközök hiányára vezethető vissza [96] részben szándékos módszertani következmény. A nyílt modellre általában, a Linux disztribúciók esetében pedig különösen igaz, hogy nagyon sok a függőséget tartalmaznak[97]. A

[3]Coupling: a programkódban található objektumok függőségét mutató mérőszám. Az erősen függő (highly coupled) kód nehezen módosítható, mert a változtatás sok más objektum működésére hatással lehet.

csomagok közötti bonyolult függőségi rendszer ellehetetleníti a különböző kiadások közötti kompatibilitást[98, 99], az egy adott verzión futó bináris egyáltalán nem biztos, hogy fut a másikon. (Ugyanez igaz egy szoftveren belül a komponensek tekintetébe is, igaz ezzel a jelenséggel a végfelhasználó csak ritkán találkozik). Ennek oka a magas szintű komponens újrahasznosításban keresendő[61, 92, 100]. A meglévő komponenseket (bináris programkönyvtárak vagy osztályok) a teljes projekt nagyon sok része használja, viszont a komponensből lehetőség szerint csak egyetlen verziót integrálnak, melynek eredményeképpen jóval kisebb, konzisztensebb, ugyanakkor az ABI/API[4] eltérések miatt sokkal kevésbé kompatibilis rendszerhez jutnak. Ebből kifolyólag egyetlen FLOSS sérülékenységnek igen széles körű hatása lehet [101].

Amikor egy komponens nem illeszkedik tökéletesen gyakran egyszerűen megváltoztatják[102] és helyi módosításként kerül a kódbázisba, ami további hibalehetőséget visz a rendszerbe (a korábban említett hírhedt openssl sérülékenység is helyi foltként került a disztribúcióba).

A függőségek kérdése azért is nagyon fontos, mert a projektek a függőségeik közt szereplő komponensprojekteket nagyon ritkán monitorozzák [94]. Ebből viszont az következik, hogy ha egy sokat használt komponensben sérülékenységet vezetnek be, az nagyon könnyen és észrevétlenül szétterjedhet, elérve és veszélyeztetve a végfelhasználó rendszereit is.

2.1.3 Minőségbiztosítás (FS-F-M)

OSSD esetén többnyire nincs formális minőségbiztosítás [103, 104] a kisebb projektek nem gyűjtenek minőségre vonatkozó empirikus bizonyítékokat[54, 105], de alkalmaznak ad hoc megoldásokat és kódminőséget javító módszereket (code readings, walk through, refactoring, peer review) [61, 85, 106]. Nagyobb projekteknél a minőségbiztosítás gyakran patchelés alapú fejlesztési megközelítés pótolja, azaz egy adott módosítás halmaz csak szigorú ellenőrzés és az ellenőrök általi digitális aláírás megszerzése után kerülhet az upstream[5] kódbázisba[107].

A minőségre implicit módon kihatnak pszichológiai tényezők is, hiszen a nyilvánosságból adódóan a kiadott kódot nagyon sokan látják, a kóddal kapcsolatos nézeteiket, a szerintük helyes megközelítést kom-

[4] API: Application Program Interface, ABI: Application Binary Interface

[5] Általános értelemben upstream kódbázisnak nevezzük a projekt által felhasznált külső kódot, programkönyvtárakat, modulokat. A verziókezelők esetében azt a verziót nevezik upstreamnek, amelyből a szóban forgó változatot készítették (forkolták).

munikálják is, ami egyrészt sokat lendít a tanulási folyamaton, másrészt bizonyos kényszerítő erőt jelent a helyes programfejlesztési módszertan és a szabványok követésére[108].

Bizonyos értelemben a zárt modellben implicit míg nyílt modellben explicit minőségellenőrzés végezhető. Míg zárt forrás esetén a minőségbiztosítás elvégzését és szintjét egy tanúsítvány bizonyítja, azaz elsősorban bizalmi kérdés, a nyílt modellben az elvégzett munka közvetlen módon, egyértelműen ellenőrizhető és mérőszámokkal jellemezhető [72, 109].

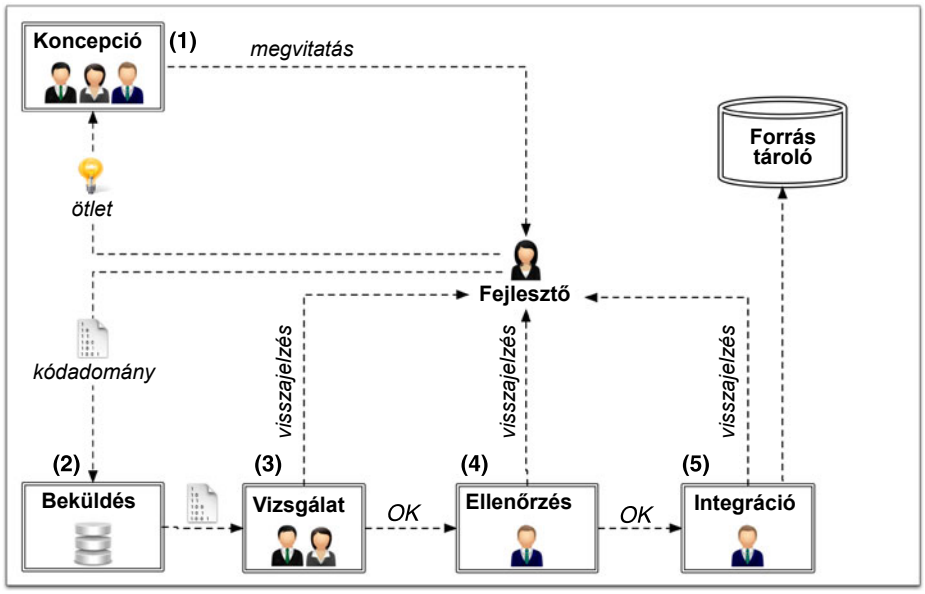
2.1.4 Kivitelezési szakasz (FS-F-V)

A FLOSS modell sok, hagyományosan elkülönülten kezelt lépést a kivitelezés során végez. A kivitelezéssel párhuzamosan futnak a béta tesztek, a tervezés, hibajavítás, más elemek viszont kimaradnak, általában nem tartják nyilván például a problémák megoldásával járó időt szemben az üzleti világban megszokott eljárással [110]. A hagyományos kivitelezési módszertan valószínűleg nem is igazán működne, az esetenként több ezres fejlesztőtábor, magas fluktuáció és az "egyszeri" fejlesztők miatt. Az átlagos közösségben töltött idő egészen alacsony is lehet (Android 65 nap, Linux 24-57 nap[6])[58]

Minthogy a fejlesztés közösségi hozzájárulásokon alapszik, ezeknek a kódmódosításoknak valahogy be kell kerülniük az upstream kódbázisba. Ez a folyamat általában szabályozott és gyakran igen komoly követelményeknek kell megfelelni[111, 112]:

1. A kódmódosítás halmaznak az aktuális fejlesztői verzióból kell származnia és alkalmazkodnia kell a projektben elvárt fejlesztési sztenderdekhez.
2. A külső féltől származó hozzájárulásokat általában valaki elemzi. E lépés célja a megfelelő minőség biztosítása, funkcionális ellenőrzés. A kódellenőrzést általában több kis csoport végzi, melyek mindegyike egy-egy területre koncentrál [113], a jó ellenőr megtalálása nem mindig könnyű, ami jelentősen megnövelheti az átfutási időt [114].
3. A kódot addig módosítják amíg el nem éri a közösség által elvárt szintet.
4. Ezek után a projekt valamely szenior tagjának jóvá kell hagynia a módosítást biztosítva, hogy az új kód hibamentes és teljesíti a regressziós teszteket;
5. végül a kód integrálásra kerül a kódbázisba.

[6]Természetesen vannak állandó fejlesztők is, az átlag az egy-két alkalommal hozzájárulók miatt ilyen alacsony.



2.1. ábra: Módosítás elfogadásának mechanizmusa OSSD esetén. (Bettenburg nyomán [58])

A teljes folyamat elég erőforrásigényes, kivitelezése gyakran meghaladja a kisebb projektek lehetőségeit, melyeknek így nehézségeket okoz a hozzájárulások menedzselése [58]. Ezen a problémán segít a közösségi fejlesztőportálok (pl. Github) pull request[7] funkciója, amely az elfogadási folyamat bürokratikus terhét részben leveszi a magfejlesztők válláról [115, 116].

Ez a többlépcsős kódelfogadási módszer fontos pillére az kódminőség biztosításának, ugyanis a tapasztalatlan fejlesztők 2-20x annyi sérülékenységet vezetnek be mint a tapasztaltabb belső fejlesztők. Érdekes módon megfigyelhető, hogy a fizetett fejlesztők is nagyobb eséllyel vezetnek be új sérülékenységet mint a közösség többi tagja [106].

A perifériáról a mag közelébe kerülők ráhatása jóval nagyobb a projektre. A magfejlesztők változtatásait sokkal nagyobb valószínűséggel fogadják el és sokkal hamarabb kapnak visszajelzést[117] is.

Általában igaz, hogy a FLOSS fejlesztésekben a változások sokkal gyakoribbak, sokkal nagyobb számú (később fel sem használt) változat, kiterjesztés, módosítás jellemzi őket ami esetenként olyan méreteket ölthet, ami már a projekt stabilitását veszélyezteti [118]. A rengeteg párhuzamosan futó változat és változtatás eredménye, hogy sokszor bizonyos részt többen is megírnak vagy módosítanak, a különféle verziók

[7]közösségi fejlesztő-portálokon alkalmazott eljárás, amely formális keretet biztosítva leegyszerűsíti a kódelfogadás folyamatát. A kérelmező saját kódváltozatából létrehozott módosításkészletet elküldi a projekt fejlesztőjének, aki azt soronként átnézheti, a változáskészlethez megjegyzéseket fűzhet, módosításokat kérhet, illetve igény szerint elfogadhatja azt.

között sokkal több a konfliktus mint zárt forrás esetében. Verziókezelő szoftverek nélkül ez a fejlesztési modell gyakorlatilag kivitelezhetetlen lenne, de még így is rengeteg a nehezen feloldható konfliktus a kódmódosítások integrálása során[119].

A nyílt fejlesztés egyik fontos velejárója a saját, alternatív változat, a fork készítésének lehetősége. Sokak szerint a forkolás joga nélkül az nyílt forrás nem is nyílt forrás, a szabad szoftver nem is szabad szoftver többé[120]. A forkolást két tényező teszi lehetővé és két fontos okból kerülhet rá sor. Egyrészt könnyű végrehajtani hiszen garantált a jogi lehetőség a másolásra, másrészt egyszerű a technikai megvalósítás a kiterjedt (D)VCS használat révén. Az indíték lehet szociális amikor a vezetéssel elégedetlen (pl. MariaDB) vagy nézetkülönbségek által megosztott közösség (pl. ffmpeg / libav) kettéválik, illetve technikai, ahol a fork a fejlesztés eszköze, szándékosan és nagy tömegben hozzák létre munkaszervezési céllal vagy egy-egy új funkció tesztelése végett (feature branch).

Az előbbit gyakran negatív felhanggal említik, ám ha a kódot a DNS lánchoz hasonlítjuk a projektek osztódása és kihalása egészséges természetes szelekciónak tekinthető[121, 122] ami segít túlélni a környezet extrém változásait (pl. akvizíció) és ösztönzi a folyamatos fejlődést.

Egyértelmű negatív hatásai az alábbiak:

- nagy számú, eltérő képességű, állapotú és kompatibilitási szintű változat kialakulása, amelyből adott esetben nehéz lehet kiválasztani a megfelelőt[123];
- duplikált erőfeszítés, az emberi erőforrások megosztása;
- valamint, hogy kompatibilitási problémákat okozhat[120].

A technikai fork egyértelműen pozitív, funkcionális jelenség, nem jár a közösség felosztásával és a modern DVCS[8] rendszerek központi funkciója (branching). A közösségi fejlesztőoldalak tulajdonképpen erre a képességre építik projektkezelési stratégiájukat. A különféle fejlesztői, feature és topic-branch forkok egymással könnyen összehasonlíthatók, szükség esetén egyesíthetők, párhuzamosan futtathatók és köztük kiválasztott kódrészretek relatíve egyszerűen mozgathatók.

Az LRE területén, különösen a kormányzati szférában lehetőleg kerülni kell a szociális eredetű forkok kialakulását mert az egyedi változat karbantartása nagy megterhelést róhat az új tulajdonosra aki így nem tudja kihasználni a fejlődő eredeti változat képességeit és biztonsági frissítéseit[124]. Amennyiben a szer-

[8]Distributed Version Control System, megosztott változáskezelő rendszerek, pl. Git, darcs, mercurial.

vezet technikai forkot használ változtatásokat lehetőség szerint vissza kell vezetni az eredeti projektbe. Az legfrissebb fejlesztői változatot általában nem javasolt éles környezetben használni az alább ismertetett kiadási jellegzetességek miatt, ilyen esetekben megoldást jelenthet a snapshotting, azaz a szervezet huzamosabb ideig egy stabilnak ítélt változatot használ és csak ellenőrzést és tesztelést követően vált a következőre. A stabil változatba csak az esetleges biztonsági foltokat vezeti vissza.

2.1.5 Tesztelési és kiadási szakasz (FS-F-T)

Gyakran hiányoznak a struktúrált tesztelési eljárások, illetve a tesztelési folyamat a projekt életének igen hosszú szakaszán át általában folyamatosan tart [85]. A kisebb projektek néha egyáltalán nem alkalmaznak teszteket, ezt teljes egészében a közösségre hagyják[125–127]. Nem minden projekt alkalmaz tesztelési dokumentációt, tesztelési keretrendszer pedig csak kivételes esetben jelenik meg [128]. A legjellemzőbb alkalmazott tesztelési eljárás a unit tesztek írása [104]. Takasawa kimutatta, hogy a megjegyzések és a tesztek fedettsége korrelációban áll a tesztek eredményeivel, így tesztek sikeressége futtatás nélkül a metrikákból becsülhető[125].

A kiadások ütemezésénél megfigyelhető, hogy előszeretettel használnak minél hamarabbi és gyakoribb kiadásokat[98] illetve, hogy a kiadás, tesztelés, kódellenőrzés és karbantartás egyidőben, párhuzamosan folyik [72]. Jó minőségű szoftver előállításához hosszas tesztelési időszak szükséges, ami látszólag ellentmondásban áll a nyílt modell gyakori kiadási ütemezésével[129]. A párhuzamos munkavégzés és a nagy számú folyamatos teszt következtében [61] a legújabb funkciót még nem tartalmazó stabil változatok megbízhatósága végső soron egyáltalán nem marad el az üzleti modellben elérttől. A gyors ütemezés célja, hogy minnél hamarabb minnél több emberhez eljusson a termék, ne csak a fejlesztők nézzék át a kódot, ezáltal a hibák gyorsan felszínre kerülhessenek. A felhasználói bázis mérete valóban korrelációban áll a hibajavítás minőségével[130].

A gyors kiadási ütemezés általános módszertana a következő[131]:

- Tervezés
- Megvalósítás
- Fagyasztás (feature freeze): a fejlesztést egy adott állapotban rögzítik és a hibajavításra koncentrálnak [132].

- Ütemezés: Viszonylag kevés projekt alkalmazza de elengedhetetlen az időhöz kötött ütemezés esetében, ami elsősorban a szponzorált nyílt forrás sajátja. Más projektekben a kiadások ütemezése alapvetően funkcionális, azaz az új funkciócsoport megjelenéséhez köthető[85].
- Mérföldkövek meghatározása: Nem minden projektnél található meg és általában csak lazán megfogalmazottak. Semmilyen garancia nincs, hogy azokat el is érik, inkább csak tájékoztató jellegűek [133].
- Határidők: sok projekt határoz meg határidőket, amiket nem mindig sikerül tartani, hiszen a kiadásért felelősszemélynek nincs befolyása a fejlesztők felett.
- Fordítás különféle architektúrákra: előnyös lehet, ugyanis bizonyos hibák csak bizonyos architektúrákon kerülnek elő [133].
- Felhasználói teszt: a kiadások legfőbb előnye a visszajelzések gyors és közvetlen begyűjtése a tesztverziót használó (rendszerint jól képzett) felhasználóktól. Még a kiterjedt tesztelési eljárásokat használó fejlesztők is úgy tartják, a leghasznosabb visszajelzések a felhasználóktól származnak.
- Kiadási ellenőrző lista: sok projekt használ kiadási feladatlistát, ami biztosítja hogy egyetlen lépés sem marad ki.
- Kiadás minősítés: kevés projekt rendelkezik formális kiadás-minősítő eljárással, a minősítés informálisan levelező-listákon és fórumokon zajlik.

A nyílt modell hibajavító képessége meglepően jó, és a várakozásokkal ellentétben nem függ a projekt méretétől, azaz a nagyobb projektek éppen olyan jól képesek elvégezni a korrekciós feladatokat[134].

2.1.6 Eszközhasználat (FS-F-E)

A nyílt modell jellemző eszközöket használ, ezek döntő többsége fejlesztői eszköz [135]. A nyílt modell sok eszköze átszivárgott a zárt fejlesztésbe is, így az eltérés ma már nem kiugró. Általában igaz, hogy a közösség szinte kizárólag nyílt fejlesztésből származó eszközöket alkalmaz, erősen jellemző a DVCS használata [136, 137] amelyek kiváló minőségűek. Jó példa az eredetileg a Linux kernelhez fejlesztett git verziókövető rendszer világméretű térhódítása. Zárt eszközök használata általában nem támogatott, hiszen ezzel korlátoznák a potenciális csatlakozók körét, ami viszont a zárt és nyílt fejlesztések együttműködésében zavart okozhat [110].

A nyílt fejlesztőeszközök jó minősége nem véletlen, hiszen klasszikusan teljesíti a fejlesztő egyben felhasználó is elvet, amely a nyílt fejlesztések egyik fő motiváló ereje.

A fejlesztőeszközök terén is igaz, hogy nem dominál a felhasználó-centrikusság, azaz, az eszközök technikai szempontból rendkívül jó minőségűek és széles képességkészlettel rendelkeznek, használatuk egyáltalán nem biztos hogy felhasználóbarát vagy kényelmes (ez természetesen szubjektív).

Mind a hibakeresés mind a fejlesztés során gyakori a kódrészletek cseréjét lehetővé tevő alkalmazások (pl pastebin) használata[138] illetve a kiterjedt információ csere fórumokon és kérdezőoldalakon (pl. Stack Overflow). Mára ez a különbség is elmósódni látszik, mert a zárt fejlesztésekben is kiterjedten alkalmazzák ezeket a módszereket.

2.1.7 Összefoglalás

Összességében a FLOSS fejlesztési modell a zárt modelltől erőssen eltérőnek mutatkozik, gyakran olyan jellemzőkkel, amelyek a zárt fejlesztés során kritikus hiányosságnak számítanának, ugyanakkor kiterjedt tesztelés és alternatív kódelfogadási módszerek révén végső soron mégis jó minőségű kódot képes előállítani. Megállapítottam, hogy a két modell az utóbbi években folyamatosan közeledik, mindkettő átvett a másikból bizonyos elemeket, de a különbségek még mindig elég jelentősek ahhoz, hogy a kooperáció során nehézségeket és így biztonsági kockázatot jelentsenek.

2.1. táblázat: A F kategóriában azonosított problémák

kód	szint	leírás	sajátosság
SF01	1	Nincs formális tesztelés, ami megnehezíti a minőség-ellenőrzést	FS-F-T
SF02	1	Nem piac vezéreltek a követelmények, a biztonság nem feltétlen szempont, amire nincs ráhatásunk.	FS-F-K
SF03	1	Nincs formális biztonsági tervezés az informális követelményeket nehéz minősíteni.	FS-F-K
SF04	3	A kódbázisba sérülékenységet vezetnek be, ami az upstream változaton keresztül eléri a szervezet kódbázisát.	FS-F-P
SF05	1	Hagyományos minőségbiztosítási elveknek nem felel meg, emiatt nehéz összehasonlítani, értékelni.	FS-F-M

kód	szint	leírás	sajátosság
SF06	1	A beszállító vagy közösség nem monitorozza saját komponens projektjeit. Az azokban bevezetett sérülékenység a terméken keresztül a szervezethez is eljut.	FS-F-P
SF07	3	A szervezet forkolja az eredeti kódbázist, de nincs erőforrása karbantartani azt.	FS-F-V
SF10	3	A komponensek közötti szoros függőségek miatt a komponens frissítése számos részkomponens frissítését vonja maga után, ami ütközéshez és hibákhoz vezethet.	FS-F-P
SF11	3	A kompatibilitás érdekében helyileg módosított komponens sérülékenységet vezet be.	FS-F-P
SF12	4	A fejlesztőtábor kicsiny mérete lassítja vagy ellehetetleníti a hozzájárulásaink időben történő integrálását.	FS-F-V
SF13	2	A fejlesztői vagy tesztelés alatti állapot használata növeli a sérülékenységek esélyét	FS-F-T
SF14	3	A szervezet fejlesztőeszközei nem kompatibilisek a projektben használtakkal, ami lassítja az együttműködést	FS-F-E

2.2. táblázat: A F kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JF01	4	A szervezet részt vesz a projekt fejlesztésében, így biztosítva a számára kedvező célok elérését.	SF01, SF02, SF03, SF05, SF12, ST02
JF02	4	A kód-hozzájárulásokat a projektre vonatkozó formai és minőségi szabályok betartásával, lehetőleg egy ismert közösségi tag segítségével kell bevezetni.	SH07, SK01, SK03, SK04
JF03	1	Közvetlenül, kvantitatív módon ellenőrizhető a minőség szintje.	SF05, ST05
JF04	3	Lehetőség van saját alternatív (forkolt) változat létrehozására.	SF07, SF11
JF05	2	Egy adott állapot rögzítésével (snapshot) elkerülhető a folyamatos változás okozta problémák egy része.	SF04, SF06, SF07, SF10

2.2 Gazdasági és társadalmi hatás (FS-G)

A nyílt forrással kapcsolatos publikációk gyakran említik a jelenség gazdaságra és a társadalomra gyakorolt pozitív hatásait. A fejezetben azt vizsgálom ezeknek a tényezőknek lehet-e valamilyen közvetett vagy közvetlen hatása a biztonságra.

2.2.1 Gazdasági hatás (FS-G-G)

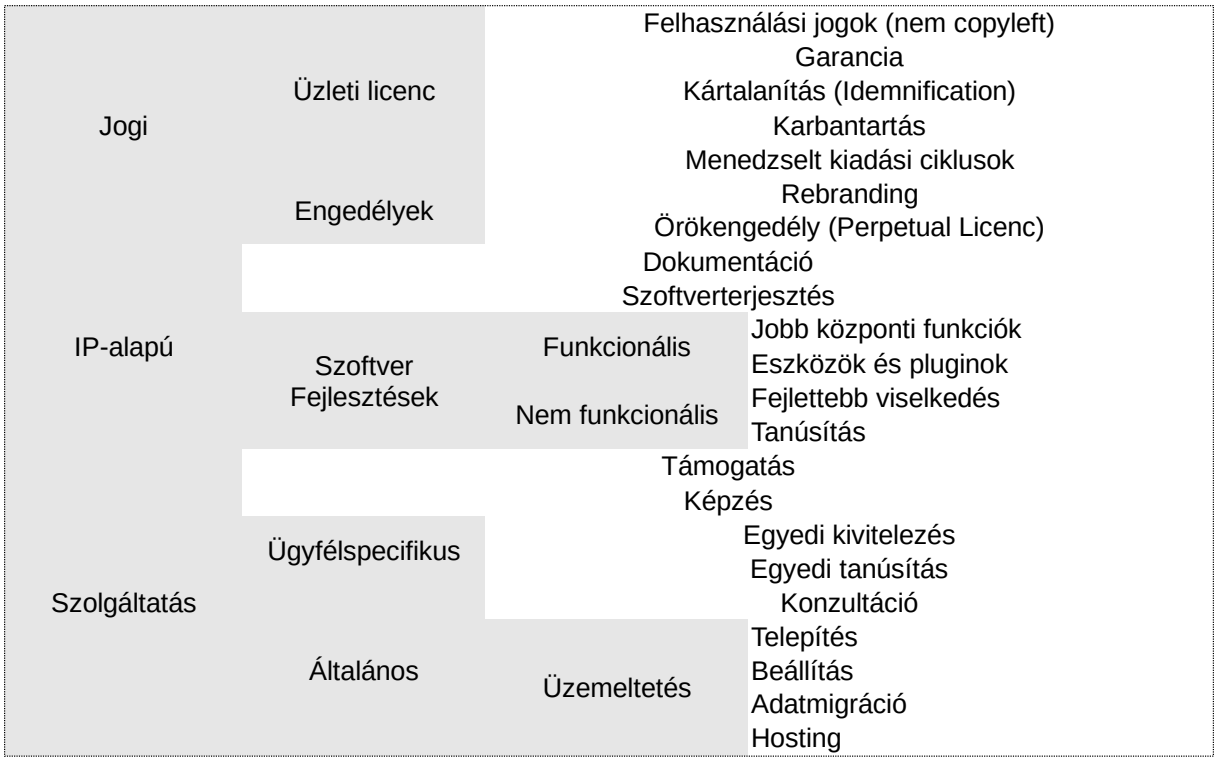
A FLOSS egyedi gazdasági sajátosságai alapján nem feltételeztem, hogy követlen biztonsági hatásokat lehet azonosítani, mivel azonban a források előzetes feldolgozása során egyértelművé vált, hogy vannak ilyenek jellegzetességek a teljesség érdekében nem lehetett ezt a kategóriát sem kihagyni. Mint kiderült, indirekt módon az itt azonosított sajátosságok ténylegesen befolyásolhatják a LRE biztonságát.

A FLOSS gazdasági kölcsönhatásai résztvevő fél alapján két csoportra oszthatók. A FLOSS üzleti szereplőkkel kapcsolatos kölcsönhatásaira és a társadalom egészét érintő hatásokra. E két hatás néhány aspektusában hasonlít ugyan, de egyáltalán nem azonos.

Számos nyilvánvaló előnye[139] ellenére a vállalatok kezdetben tisztán financiális okokból használták a szabad szoftvereket. (Valójában mind a mai napig ez a legfőbb motiváló erő [140, 141]). Csakhogy ez a modell nehézkesen működik, hiszen minden racionális szereplő érdeke azt diktálja, hogy "ingyenélőként" használja a terméket és értékes erőforrásokkal már ne járuljon hozzá [142, 143]. Ilyen szemlélettel nem is igazán érthető mi visz rá egy gazdasági szereplőt arra, hogy "ingyen" fejlesszen.

A közösségi célokba fektetett energia csak úgy térülhet meg, ha a cégek kiegészítő termékekkel végső soron profitot termelhetnek általa. (Épp emiatt tartják általában a kevésbé kötött licenceket előnyösebbnek [144]).

Így nézve a közösségi fejlesztés csak első látásra tűnik rossz befektetésnek, valójában nagyon is konkrét gazdasági előnyökkel járhat. A vállalkozás kérhet pénzt jogátruházásért, támogató tevékenységéért, egyedi fejlesztésekért, különleges szoftverelemekért, támogatásért, képzésért, üzemeltetésért anélkül is, hogy a forráskód kizárólagos tulajdonjogát birtokolná. Az gyakorlatban alkalmazott üzleti lehetőségeket Weikert nyomán 2.2. ábrán látható táblázatban foglaltam össze.



2.2. ábra: Nyílt fejlesztés gazdasági lehetőségei (Saját szerkesztés, Weikert nyomán [30])

Természetesen a haszon realizálódhat más, indirekt módon is, például korunk egyik vezető "valutájában" információ formájában vagy presztízsértékként. Minthogy a legjobban képzett fejlesztők jelentős része maga is részt vesz valamilyen FLOSS projektben a közösségben részt vállaló cégek könnyebben megszerezhetik és megtarthatják őket[145].

Idővel – elsősorban az IT szektorban – a gondolkodásmód megváltozott. Azok a cégek amelyeknél hosszabb távú stratégiai célokat is megfogalmaztak, világosan látták, hogy sokkal többet nyerhetnek egyszerű rövid távú gazdasági előnyöknél. Végül a folyamat öngerjesztővé vált, megkezdődött a FLOSS üzletiesedése [146]. Ma már a FLOSS a szoftverpiac egyik vezető hajtóereje, amely figyelemre méltó módon növekszik [12], egyre elterjedtebb a kifejezetten nyílt forrású stratégia megfogalmazása, a cégek fontosnak tartják a nyílt forrású együttműködést sőt megszületett a cégen belüli, FLOSS módszerekkel való munka fogalma, az "Inner Source" [147]. A FLOSS gazdasági előnye mérhető. A top1000 vállalat esetében korreláció mutatható ki a tőzsdei siker és a FLOSS használat között [7].

A forráskód megnyitásának kényszerítő ereje is van. Ha ugyanis egy piaci szereplő egyoldalúan megnyitja a forráskódot ellenfelei könnyen rosszul járnak. Asundi gazdasági modellszimulációjának egyik eredménye az volt, hogy a versenytársnak is érdemes ilyenkor megnyitna a forráskódot, ha viszont mindketten

megnyitják a forrást, valamivel kevesebb haszonhoz jutnak, semmiképpen nem nőhet a nyereségük. Ami előrevetíti a modell másik eredményét, miszerint ha a teljes gazdaság nyílt forrású lenne, az korántsem lenne optimális. Valójában a nyílt és zárt modell együttélése az, ami a leghatékonyabb, de ez csak akkor valószínű, ha a nyílt változatból fontos képességek hiányoznak vagy ha a üzleti szoftver kiegészítő előnyökkel bír [148]. A nyílt forrás tehát serkenti a versenyt de akár egyfajta gazdasági fegyver is lehet, amivel napjaink nagyvállalatai minden valószínűség szerint élnek is.

Ilyen okból vagy sem, számos nagyvállalat adományozott, azaz tett nyílttá korábban zárt forráskódot [149], ami igen jó – mondhatni egyedülálló – lehetőség a nyílt és zárt fejlesztési modell és kódminőségének összehasonlítására.

Egy nemzet gazdasága számára több okból is kedvező lehet a FLOSS fejlődése és elterjedése [108, 149–151]:

- segíthet fejleszteni a nemzeti szoftveripart;
- segít kontrollálni a kormányzat licenckölségeit, csökkenti a függőséget;
- növeli a nemzeti információbiztonságot;
- általános monopolellenes hatása, csökkenti az ország importfüggőségét;
- segít elkerülni a hálózathatáson[9] alapuló piaci egyenlőtlenségeket;
- a szabad szoftver erős fegyver az üzleti szoftverek ártárgyalásakor;
- illetve enyhíti a kalózkodás problémáját.

A globális szoftveripart az információgazdag nagyvállalatok uralják, amelyek három jogi konstrukción keresztül tartják irányítás alatt a piacot: a copyright jogok, szabadalmak és az üzleti titkok jogi védelme. Ha az információ-áramlás e három gátját eltávolítanánk akkor megszűnne a szoftverfelhalmozás késztetése és lehetősége. Következésképpen a piacon nem az lenne uralkodó, aki több szoftvert tud felhalmozni, hanem aki jobb fejlesztőket tud magához csábítani és megtartani. A nyílt forráson alapuló fejlesztés lényegében ezt a célt valósítja meg. Úgy tűnik tehát – legalábbis a szoftveripar terén – a FLOSS alapú gazdaság egyáltalán nem katasztrófa, kialakulása teljesen természetes, talán elkerülhetetlen is [152].

[9] Az ökonómiában és üzleti világban használatos fogalom azon pozitív hatás leírására, amikor a termékhez vagy szolgáltatáshoz csatlakozó minden további felhasználóval mindenki számára tovább nő a termék értéke. Ha a hálózathatás érvényesül, a termék teljes értéke a felhasználók számával együtt növekszik. Klasszikus példák a telefon és a szociális hálózatok. Forrás: Wikipédia

Felfoghatjuk úgy is, hogy a FLOSS átmenet képez a privát befektetési modell és a közösségi tevékenységi modell között. Az első esetben privát befektetés fedezi az innovációt a megtérülését pedig IP törvények védik. A második esetben az innovációt ajándékként kapja a közösség, azt bárki korlátozás nélkül használhatja. A FLOSS innovációs modell esetén az emberek azért csatlakoznak, mert a személyes haszon sokkal nagyobb a közös munkában részt vállalók, mint az "ingyenélők" számára [41, 153].

A könnyen elérhető kész komponensek jelentősen lerövidítik a fejlesztési időt így a fejlesztő cég a lényegi elemekre tud koncentrálni, ami versenyelőnyt jelent [75]. A FLOSS tehát stimulálja az innovációt[121, 154], segíti a technológiai fejlődést [155] és helyzetbe hozza a kisvállalkozásokat [156]. Mindamellett a FLOSS gyengítheti a szoftverfejlesztés hagyományos motivációs rendszerét[157] hiszen gátolja a direkt financiális megtérülés lehetőségét, így FLOSS esetén fontos lehet más motivációs struktúrát keresni (FS-K-R).

Lin szerint amikor egy nyílt és egy zárt termék harcol valamely piacért és a zárt termék jelentős hálózathatásból származó előnnyel indul, árcsökkentéssel teljesen kiszoríthatja a FLOSS terméket a piacról. A FLOSS szoftverek léte csökkenti az üzleti változatok profitját, a különbség pedig a felhasználónál csapódik le [158]. A FLOSS támogatása tehát nemzetgazdasági érdek is lehet.

A nyílt fejlesztés globális biztonságnövelő hatása talán nem feltétlen egyértelmű. A terméktulajdonságokról szóló fejezetben (FS-J-K) részletesen bemutatom, hogy a forrás nyíltsága önmagában nem okoz szignifikáns növekedést a szoftverbiztonságban. A terhek megosztása az, ami komoly eredményekkel kecsegtet a biztonság terén. Egyetlen sérülékenység feltárásának becsült erőforrás igénye 5000 emberóra, ami jelentős terheket ró a kizárólag belső fejlesztést végző cégekre. A lehető legmagasabb biztonsági szintet nem a tökéletesen biztonságos rendszer megalkotásával lehet elérni – hiszen ilyen, a triviálisnál bármivel összetettebb rendszer nem létezik – hanem a rendelkezésre álló szűkös erőforrások helyes felhasználásával [71].

Az Unió figyelmét sem kerülte el a terület problémái és a benne rejlő lehetőség. A 2014-ben lefektetett Nyílt Forrású Szoftver Stratégiában a következő stratégiai célokat fogalmazza meg:

- egyenlő feltételek a beszerezéseknél;
- részvétel a közösségben;
- jogi kérdések tisztázása;

- a EK által fejlesztett szoftverek kódjának megnyitása;
- átláthatóság és jobb kommunikáció.

Ezekből az elvekből egyértelműen látszik a nyitás szándéka, amely az előnyök felismerésből ered. Belátható, hogy a gazdasági racionalitás ugyan fontos komponense a FLOSS fejlődésének, de a biztonságra gyakorolt pozitív hatások csak akkor érvényesülhetnek teljes mértékben, ha a társadalom tagjai – elsősorban a döntéshozók – jobban megértik a jelenséget, annak előnyeit és hátrányait.

2.2.2 Társadalmi hatás, tudati dimenzió (FS-G-T)

A biztonság nem egyszerűsíthető le tisztán technikai kérdésekre mindig figyelembe kell venni az emberek gondolkodását, hozzáállását, pszichológiai tényezőket is[159]. Nem lehetetlen tehát, hogy a nyílt modellnek a tudati dimenzión keresztül is vannak biztonsági hatásai.

A FLOSS megítélése nem homogén a társadalomban, akik jobban ismerik pozitívabb véleménnyel vannak róla, de általában sok a bizonytalanság a jelenséggel kapcsolatban [123, 160]. A Stack Overflow kommunikációjának elemzése alapján a FLOSS erősen jelen van a fejlesztői gondolkodásban [161] az alkalmazottak számára pedig általában vonzó és előremutató [162] ha a cég nyílt modellben gondolkodik. Az is megfigyelhető, hogy minél nagyobb klánkontrolt[10] gyakorol egy gyártó és minél több tudást vonzz be, annál nyitottabbá válik [163].

A felhasználók részéről továbbra is megfigyelhető az ellenállás [164, 164], az erős hálózathatás pedig néhány területen tovább gyengíti a FLOSS elterjedését [7] (van persze ahol éppen erősíti). Gyakran hangoztatott akadály a nehéz használhatóság [165].

A nyílt modell termékeit elfogadók körében kezdetben általában financiális okok domináltak, a politikai és szociális felelősség nem játszott szerepet [166]. A gazdaságosság, az azonnali haszon általában ma is nagyon fontos, a módosíthatóság és szabadság háttérbe szorul[167] így a vezetőség fejében torz kép alakulhat ki.

Komoly kulturális szakadék figyelhető meg a menedzsment és a OSSD fejlesztők között, a menedzsment úgy érzi, nincsenek felelősök, nincs kit hibáztatni a kudarcokért, aggódnak a rejtett költségek miatt [168,

[10] irányítás informális módja, ahol az irányított és az irányító közötti függést a közös értékek és célok közötti szakadék szűkítésével érik el.

169], továbbá, ha nincs a fejlesztés mögött egy “komoly cég” az egész amatőrök játékának tűnhet [139]. A biztonság-kritikus területeken ez a jelenség különösen erőteljes [70, 170].

A szervezet fejlesztői viszont – akik közelebbi kapcsolatot ápolnak a FLOSS rendszerekkel és kedvelik az innovációt – úgy érezhetik, hogy falakba ütköznek, esélytelen meggyőzniük a vezetést, ezért aztán elkezdik “fű alatt” használni[171]. A FLOSS használat tehát lehet rejtett (konkrét példák vannak rá például az energia szektorban [172]) ami oda vezethet, hogy a menedzsment – annak biztos tudatában, hogy FLOSS rendszert nem használnak – rosszul méri fel a kockázatokat, a FLOSS biztonságos használatához szükséges szabályozás pedig elmarad.

Gyakori probléma a tapasztalatok és könnyen elérhető referenciák hiánya [173] az ellenérzést felnagyíthatja a támogatás területén látszólag tapasztalható hiányosságok. Az eltérő támogatási modell meg nem értéséből fakadóan a vezetők nem is ismerik fel a FLOSS-t hatékony alternatívaként [143, 172]

Magas szintű összefogásra lenne szükség ahhoz, hogy követendő minták, lehetőségek és kockázatok mindenkihez könnyen eljuthassanak. (Európában szerencsére létezik is ilyen jellegű kezdeményezés JoinUp néven[11].) Az egységes forráshasználat érdekében nagyon előnyös lenne egy közös kormányzati forráskód-kezelő rendszer kialakítása [136].

A másik terület ahol a kormányzat sokat segíthet a lehetőségek és kockázatok megértésében az oktatás. Az érzékelt használati egyszerűségre a képzés van a legnagyobb hatással [174] így ha a fiatalok találkoznak nyílt verziókkal, már sokkal kevésbé találják használhatatlannak azt. Az oktatási intézményekben viszont általában akkor is üzleti verziókat oktatnak ha van nyílt alternatíva, mivel a kialakult vélemény az, hogy a piacon erre lesz szükség[78] ami végül tovább erősíti azt a társadalmi hatást, hogy ez valóban így is legyen. Ha az elemi iskoláktól kezdve csak zárt üzleti megoldásokat látnak az emberek, az a képzet alakulhat ki bennük, hogy a technológiát csak ezen a módon tudják megbízhatóan használni[175].

Az informatika képzésbe ideje kezd beszivárogni a nyílt modellel kapcsolatok tudnivalók oktatása (vannak nyílt forrással kapcsolatos kurzusok többek közt a UNH-en [176] hazánkban pedig a Műszaki és Gazdaságtudományi Egyetemen vagy Számalknál) de a közösség-építéssel, közösségi együttműködéssel kapcsolatos tudnivalókat nemigen oktatják. A fejlesztő-közösségekben résztvevők jellemzően autodidakta módon tanulnak, ugyanakkor – elsősorban a fiatalabb korosztály – előnyben részesítené a formális képzést [177].

[11] https://joinup.ec.europa.eu

2.2.3 Összefoglalás

Az elemzett publikációk alapján megállapítható, hogy a nyílt forrásnak mint jelenségnek jelentős gazdasági és társadalmi haszna van. A tisztán nyílt forrású gazdaság ugyan nem lenne hatékony - így nem is kívánatos - de a megfelelő arányú együttműködés jelentős előnyökkel bír. A FLOSS Állami és nemzetközi szintű támogatása tehát előnyös a teljes társadalom számára, a munkamegosztás révén pedig a szoftverek biztonsági szintje nő.

Az üzleti és nyílt modell kulturális különbségei valamint a berögzült szokások folytán a FLOSS hatásai és az OSSD módszerei ritkán szerepelnek az oktatásban, bár az utóbbi években pozitív változás érzékelhető e téren. A megfelelő egyensúly megteremtése érdekében a FLOSS specifikus oktatásra kiemelt figyelmet kellene fordítani.

A vezetés és a kivitelezést végzők FLOSS-al kapcsolatos hozzáállásából adódó különbség oda vezethet, hogy a kivitelezők rejtett módon alkalmazzák a FLOSS komponenseket, ami szabályozás híján biztonsági problémákhoz vezethet.

2.3. táblázat: A G kategóriában azonosított problémák

kód	szint	leírás	sajátosság
SG01	1	Rejtett FLOSS használat miatt rosszul mérik fel a kockázatokat és nem alkalmazzák a szükséges védelmi eljárásokat.	FS-G-T

2.4. táblázat: A G kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JG01	1	A FLOSS állami vagy nemzetközi szinten kell támogatni, mert fellendülése a munkaoptimalizáció révén növeli az információbiztonságot.	SJ01
JG02	2	A nyílt modell kockázatainak és lehetőségeinek oktatása szerepeljen a megfelelő oktatási intézményekben.	SG01
JG03	3	Országos vagy európai egységes forráskód kezelő rendszer kialakítása szükséges.	SF05, SH08, SJ01

2.3 Felhasználás (FS-H)

Ebben a kategóriában a felhasználó szemszögéből vizsgálom a FLOSS rendszereket. Ide soroltam minden olyan tevékenységet vagy eljárást, amit a FLOSS esetén másképpen, speciális körültekintéssel kell végrehajtani illetve, amely kifejezetten a felhasználókat jellemző tulajdonság.

A felhasználó esetünkben nem feltétlenül végfelhasználót jelent, ide tartoznak a belső fejlesztők, rendszerintegrátorok, disztribútorok is amennyiben az adott sajátossággal felhasználóként kerülnek kapcsolatba.

2.3.1 Széles választék, egyedi minősítő rendszer szükségessége (FS-H-M)

Feltételezésem szerint a FLOSS rendszerek minősítésére nem tökéletesek a meglévő minősítési eljárások. A zárt forrású megoldások minősítéséhez készült keretrendszerek módszertana és követelményrendszere nem képes megfelelően lefedni a FLOSS vagy akár a szabad szoftver minden aspektusát.

Akár komplett rendszerről akár komponensekről van szó, a terjesztő hiánya komoly hátrány jelenthet, mivel a COTS beszállítók értékeléséhez használt sztenderdek itt nem használhatók [178]. Az igen népszerű TCO[12] alapú becslések alkalmazása korlátozott, mert a meglévő rendszerek csak nagyon költségesen illeszthetők a nyílt forráshoz, ami miatt a vállalkozások gyakran bele se fognak [173], nem beszélve róla, hogy sem a TCO sem a komplexebb FBV[13] nem alkalmas önmagában arra, hogy egy termék minőségéről részletes képet adjon [179]. A szabad szoftverek elemzéséhez egyedi SRGM[14] modellek és rendszerek szükségesek[180], de a megfelelő változat kiválasztása itt sem könnyű. Az elemzés – hosszabb távú projektek esetén – SCRM szempontjából különösen fontos, hiszen ha az OSSD közösség nem elég stabil, a támogatás megszűnhet, holott bizonyos szektorokban évtizedekig kellene támogatást nyújtani[181, 182]. A választás nehézségeinek szemléltetéséhez és egy lehetséges választási módszertan iránt érdeklődőknek Ullah munkásságának tanulmányozását javaslom [183].

A nyílt forrás egyik jellegzetessége, hogy ugyanazon feladatkörhöz számtalan, egymáshoz nagyon hasonló változat készül (FS-F-P). Következésképpen nem nyilvánvaló, hogy hogyan tudjuk kiválasztani a megfe-

[12]Total Cost of Ownership: valamennyi direkt és indirekt költség becsült értéke a termék beszerzése és teljes használata során
[13]Full Business Value: a befektetés teljes értéke, rendszerhatékonyság, üzleti hatékonyság figyelembevételével]
[14]Software Reliability Growth Model.

lelő változatot, hogyan különböztetjük meg a jó és rossz projekteket [104, 184]. Hauge kutatásaiban a válaszolók egyértelműen bizonytalanok voltak a megfelelő FLOSS megoldás kiválasztását illetőn, sőt a lehetőségek nagy száma problémát jelenthet annak eldöntésében is, hogy a meglévő összetett rendszer helyettesítésére létezik-e egyáltalán FLOSS alternatíva vagy sem [162, 185].

Kimutatható, hogy a nyílt forrás esetében a felhasználó óriási mennyiségű gyenge minőségű változattal találkozhat. Bár a csúcshoz közeli változatok minősége felveszi a versenyt, esetenként meg is haladhatja a zárt forrású megfelelőik képességeit [186], a gyenge változatok torzíthatják az összehasonlítást és megnehezítik a választás.

Nyílt forrás esetében maga a minőség fogalom is átértékelődhet. Garvin négyféle minőséget definiál. A transzcendens minőség kizárólag személyesen tapasztalható meg, a termék-minőség diszkrét mérhető termék karakterisztikákon, a felhasználói-minőség szubjektív fogyasztói elégedettségen alapszik, a gyártói-minőség a specifikációnak való megfelelést jelenti, míg az érték alapú minőség a specifikációknak megfelelést helyezi előtérbe egy adott elfogadható áron. A hagyományos szoftverfejlesztés esetén a felhasználói elégedettség (elégedetlenség) kezelése a problémamegoldással szorosan összefonódik, költséghatékonysági szempontok miatt csak együtt, meghatározott időközönként más foltokkal, javításokkal együtt lehet azt teljesíteni. Következésképpen a zárt forrás esetén Garvin értékszemléletű minőség nézőpontja az uralkodó, ezzel szemben a FLOSS minőségének felfogása a felhasználó alapú minőségi nézőponthoz áll közelebb. Azaz, míg az első esetben a felhasználó az ár függvényében vizsgálja az elérhető képességeket és minőséget, a FLOSS esetén a minőség önálló tényezőnek számít, a felhasználó aszerint választja ki a legmagasabb minőséget, hogy mi elégíti ki leginkább személyes elvárásait [187].

Szerencsére léteznek olyan projekt indikátorok, amelyek segítségével becsülhető a projekt érettsége (maturity level) illetve a későbbi fejlődés iránya. Ilyen lehet például az új kiadások vagy a nyitott ügyek számának változása, amelyek statisztikai módszerekkel elemezhetőek [188]. FLOSS esetén egyedi paraméterek is elemezhetők, úgy mint a felhasználói közösség hosszú és rövid távú léte, a megfelelő méretű felhasználótábor, licenszelés, felhasznált programozási nyelv egysége, komplexitás, nyelvek támogatottsága, szponzorok száma valamint az elérhető útmutatók minősége és száma [189]. Ezek a más esetben elérhetetlen vagy csak nehezen hozzáférhető információk segíthetnek felbecsülni a projekt állapotát.

A vázolt problémákat felismerve Németország, Malajzia és Ausztrália kormányzata korábban is készített

olyan átfogó irányelveket, modellt amelyet a szervezetek az FLOSS értékeléshez felhasználhattak [164] később pedig több uniós projekt is indult a jellemző metrikák nyilvántartására és analízisére. Ilyen, Európai Közösség által finanszírozott projekt volt a FLOSSMetrics[190], OSSmeter[15] és a MARKOS[16].

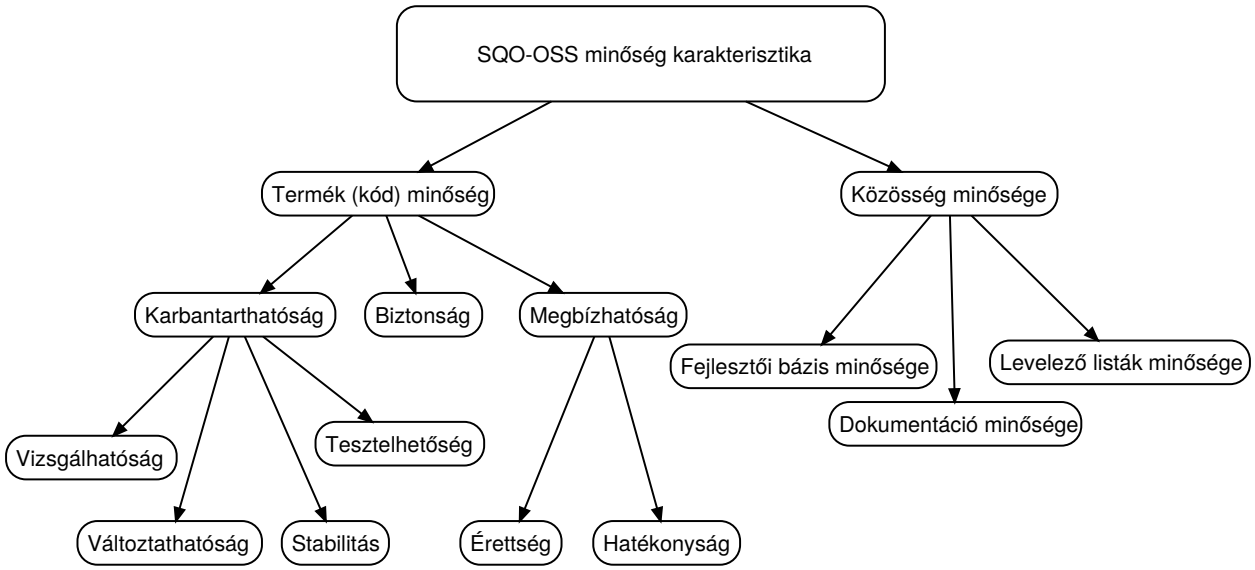
A terület fontosságát mutatja, hogy napjainkban már jó néhány minősítő keretrendszerből választhatunk[191]. Illusztrációképpen a fig. 2.4 a QuESo megbízhatósági modell eredményének vizualizációját mutatja a GNOME projekten. Néhány további lehetséges FLOSS minősítő keretrendszer:

Akronim	Minősítő rendszer	Fejlesztő/támogató	Év
QSOS	Methodology of Qualification and Selection of Open Source Software	Atos Origin	2005
C-OSMM	CapGemini Open Source Maturity Model	Capgemini	2003
OpenBRR	Open Business Readiness Rating	Carnegie Mellon West University, Spike Source, Intel, O'Reilly's	2005
E-OSS	Easiest Open Source	SIAD-Laboratory	2015
N-OSMM	Navica Open Source Maturity Model	Navica software	2004
SQO-OSS	Software Quality Observatory for Open Source Software	European Community's Sixth Framework Programme	2008

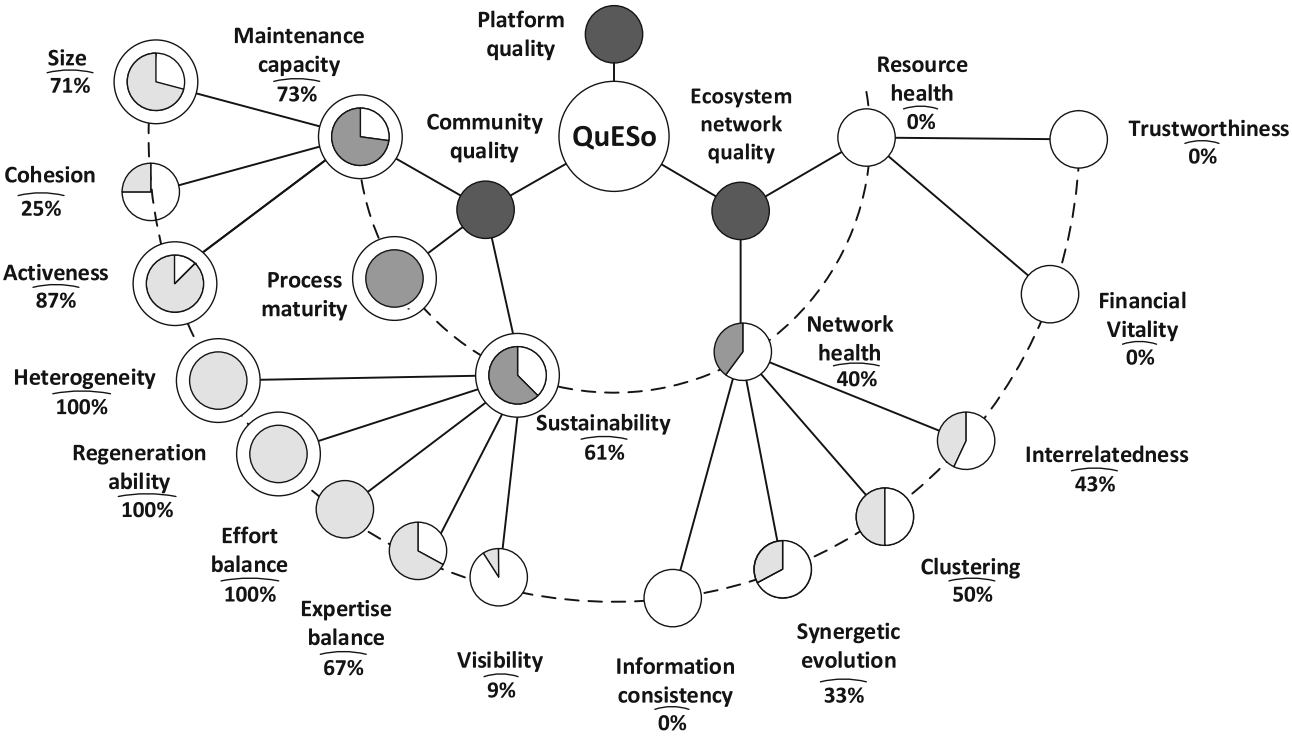
A CapGemini, QSOS, OpenBRR és SQO-OSS modellek a ISO/IEC 9126 minőségi modell irányelveit követve építik fel a keretrendszert, a QualOSS (és részben az OpenBRR) a CapGemini megoldásának továbbfejlesztései. A három szintű OpenSource Maturity Model az egyetlen amely a Capability Maturity Model[17] módszertanának irányelveit követi. [193–197]

Az általános és több egyedi [199, 200] megbízhatósági modell mellett létezik kifejezetten kockázat alapú minősítő rendszer, ilyen például Siena kockázatmodellje [201], illetve célzottan kódbiztonsági becslést végző modell mint a CodeTrust[202].

[15]Európai projekt a nyílt forrású szoftverek automatikus analízisére és mérésére (http://www.ossmeter.org/)
[16]EK Projekt: Global, integrated and searchable open-source software (https://www.markosproject.eu/)
[17]Capabilit Maturity Model https://en.wikipedia.org/wiki/Capability_Maturity_Model



2.3. ábra: SQO-OSS modell. Forrás: [192]



2.4. ábra: Gnome projekt minősítése QuESo minőségbiztosítási keretrendszerrel Forrás:[198]

A vizsgált tudományos anyagok és a szerzők egyöntetű véleménye alapján kimondható, a FLOSS rendszerek minőségének elbírálásához egyedi keretrendszerek szükségesek, amelyek figyelembe tudják venni a fejlesztési módszertanból, terjesztési eltérésekből és licenszelési különbségekből származó jellegzetességeket.

2.3.2 Fejlesztői felhasználás, integráció (FS-H-I)

FLOSS termékek integrációja jelentősen csökkentheti a fejlesztési kockázatokat amennyiben a szervezet saját erőből ilyen tevékenységet végez. A fejlesztéshez és teszteléshez szükséges idő drasztikusan csökkenhet, a közösség-alapú fejlesztésnél pedig sokan önként jelentkeznek a fejlesztői munkára [59, 203]. Az félkész FLOSS komponensek használata erőforrásokat szabadít fel, lerövidíti a fejlesztési szakaszt ami által jobb minőségű szoftverek jöhetnek létre. Ez minden szektorra igaz amelyet az információ technológiai hatásai elkezdtek átformálni, de különösen erősen érzékelhető a fogyasztói elektronikai termékek esetén[75]. A zárt forrású fejlesztési módszert alkalmazó harmadik féltől származó komponensek integrációs lehetőségei limitáltak, a FLOSS változatok lényegében tetszőleges mértékben alakíthatók igény szerint [204].

A komponens integráció két elterjed formája az úgynevezett Black-box és a White-box integráció. Az előbbinél a fejlesztő csak a bináris formát használja fel, így a kód elemzésére nincs lehetőség. Az utóbbi a forrás felhasználására utal, ami többnyire csak FLOSS komponensek esetében lehetséges [100]. White-box integráció estén lehetőség nyílik az integrált komponens átalakítására is ami sokszor jelentős előnyt biztosít ugyanakkor a szabályozott interfész megkerülésre révén bizonyos kockázatokat is hordoz.

A FLOSS felhasználásának egyik fő motiváló ereje a függetlenség ígérete. A Szervezet a szállítótól való függetlenedési törekvése során gyakran felismeri, hogy a nyílt forrást stratégiai eszközként használhatja, így a szervezet eredményeit vagy új lehetőségeit tekintve vonzóbbnak találja a FLOSS változatot [205] vagy előnyösebb alku pozícióba kerül a szállítóval szemben. Az interneten keresztül könnyen elérhető FLOSS komponensek használata ma már kétségtelenül stratégiai szerepet tölt be az iparban. [206] A nyílt forrású komponens integráció aránya folyamatosan növekszik, maga az integráció, migráció és jogi kérdések problémája viszont sok esetben nem megoldott. Egy esetben 3500 java programozó között végzett felmérés szerint a tipikus java alkalmazások 80%-a tartalmazott FLOSS komponenseket, ugyanakkor a szervezetek 76%-a nem volt képes megfelelően kezelni saját FLOSS felhasználását (2013-as adat) [207].

A FLOSS alkalmazás felhasználással ellentétben (FS-H-F) komponens integráció tekintetében nem mutatható ki statisztikailag szignifikáns különbség a kis és nagy vállalatok között [59] azaz a nagyvállalatok relatíve szívesebben használnak FLOSS komponenseket mint termékeket.

A FLOSS komponensek kiválasztásánál a COTS komponenseknél megszokott módszertan gyakran nem ad elégséges eredményt [208] és egyedi metódusokat kell alkalmazni [209]. A komponens kiválasztása tekintetében két uralkodó megközelítés létezik. Sokan egyszerűen rákeresenek a komponensekre az Interneten, míg mások – elsősorban kritikus alkalmazások fejlesztése esetén – FLOSS komponensek kiválasztására szakosodott portálokat használnak [210] vagy vállalkozások szaktudását veszik igénybe. A jó komponens kiválasztás módszere kritikus szerepet játszik a fejlesztésben, ugyanakkor az alapja gyakran szubjektív, tapasztalati úton elérhető tudás, így különösen fontos a megszerzett információk megosztása a közösséggel[211]. A kiválasztást végzők többsége egyáltalán nem használ illetve nem is ismer semmilyen formális módszertant ami a komponens kiválasztási folyamatban segítséget nyújthatna, ehelyett informális módszerekkel dolgoznak és sokszor nem is dokumentálják a kiválasztás és döntés indokait és folyamatát. [206, 212]

Pedig az integráció egyáltalán nem veszély nélküli folyamat, az integrátornak számos kihívással szembe kell néznie, a közösségben való részvételnek ugyanis ára van. Sok gondot okozhat a FLOSS komponensek közötti koordináció és figyelembe kell venni, hogy az egyes komponensek kiadási dátuma, frissítési ciklusa, fejlesztési terve eltérő lehet. A projektnek tehát fel kell készülnie saját és a közösség munkájának összehangolására. Stol kutatásaiban részletesebb felosztást alkalmaz. Az általa azonosított kihívásokat a 2.6 táblázat foglalja össze (az ebben a fejezetben nem tárgyalt tényezőket a harmadik oszlopban jelölt kategóriánál dolgoztam fel)[98].

2.6. táblázat: FLOSS integrációja során felmerülő lehetséges kihívások, Stol nyomán [98]

Kategória	Azonosított kihívás	Fejezet
Termékkiválasztás	A minőségi termék beazonosítása nehéz a bizonytalanság és a nagy számú variáns miatt	
.	Nincs idő a komponensek értékelésére	
.	A megfelelő fork kiválasztása problematikus	FS-F-P
Dokumentáció	Dokumentáció hiánya vagy gyenge minősége	FS-J-D

Kategória	Azonosított kihívás	Fejezet
.	Azonos komponens eltérő leírásokkal	FS-J-D
Közösségi támogatás és karbantartás	Függés a közösségtől támogatás és frissítések terén, a támogatás minősége nehezen irányítható, nincs helpdesk vagy technikai segítség	FS-T-T
.	Az egyedi változtatásokat karban kell tartani, ami időigényes és a jövőbeli változatokkal összeütközésbe kerülhet	
.	Az FLOSS közösséget nehéz meggyőzni a változások elfogadásáról, a részvétel drága és nehézkes. A szerkezetet nehéz irányítani ha nem vagyunk tagja a core fejlesztők csapatának.	FS-F-K
.	Bizonytalanság a termék jövőjét illetően a cég termékének vonatkozásában	
.	A közösség tagjai nagyobb beleszólást akarnak a termék tervezett funkcionalitásába	
.	A FLOSS projektbe belépés és fejlesztés plusz erőforrásokat igényel	
Integráció és szerkezeti felépítés	Visszamenőleges kompatibilitási problémák	
.	Hiányzó képességek vagy a keretrendszerhez való illesztés miatt módosításokat kell eszközölni.	
.	Inkompatibilitási problémák a meglévő rendszer és a komponensek között.	
.	Horizontális integráció.	
.	Vertikális integráció, platform vagy fejlesztői nyelv eltérése.	
Migráció és felhasználás	Konfiguráció komplexitása	FS-J-H
.	Felhasználói képzés költsége	
Jogi és üzleti kérdések	Komplex licenszelési szituációk	FS-Sz-P
.	Aggályok vagy stratégia hiánya az IP és jogi kérdésekben	FS-J-H
.	Egyértelmű üzleti modell hiánya ami elfogadható lenne az ipar számára	FS-T-Ü

Az integráció további veszélyforrása, ha az integrátor nem ismeri a felhasznált kódbázist így akaratlanul

is új hibákat vezethet be. A komponensek között bonyolult függőségi viszony létezhet, a komponensek újabb komponensektől függenek, így az azokban rejlő hibák veszélyeztethetik a felhasznált komponens biztonságát is[97]. Ezen kívül, ha a helyi módosítások nem kerülnek be az eredeti kódbázisba, azokat folyamatosan karban kell tartani, ami erőforrás igényes és további hibalehetőségeket hordoz [97, 102, 213], a komponens változásait minden egyes verzióváltáskor újra ellenőrizni kell, a fejlesztést pedig lehetőség szerint szinkronizálni kell a forrás (upstream) projekttel. A Nokia n800-as táblagépe például nagy számú (428 darab) FLOSS komponenst használt, amelyek jelentős részét (~50%) módosították. Felmerül a kérdés, a gyártó hogyan tudta biztosítani a rendszer stabilitását a módosítások visszavezetése nélkül.

Minden egyes felhasznált komponens újabb függőséget alakít ki, amely rejtett karbantartási költséget és biztonsági kockázatot jelent. A Debian szoftverkönyvtár-csomagoktól például átlagosan 6.4 másik csomag függ közvetlenül (median: 2.0) és 47.6 tranzitív módon (median: 3.0) [97]. Jól demonstrálja a jelenség veszélyeit az OpenSSL csomagban feltárt biztonsági hiba, amely helyi javításként került a Debian változatába ahol teljes két éven keresztül (2006-2008) folyamatosan jelen volt. A kriptográfiai kulcs megfelelő véletlenségét biztosító függvényt a helyi javítás megjegyzésbe tette [101, 214]. A karbantartó kapcsolatba lépett a upstream karbantartókkal, ám mivel nem fedte fel magát és terveit teljes mértékben, lényegében figyelmen kívül hagyták. A biztonsági sérülékenység 44 másik csomagra terjedt át anélkül, hogy a karbantartók vagy a fejlesztők azonosították volna a hibát.

A kriptográfiai könyvtárak sajnálatosan elterjedt black-box integrációja egyébként is aggályos, hiszen egyetlen biztonsági hibának széles körű következményei lehetnek [101].

Mint korábban erre felhívtam a figyelmet, a változtatásokat, különösen a hibajavításokat vissza kellene vezetni[181]. Sajnos ezeket a változtatásokat gyakran nehéz elfogadtatni [185, 215]. Továbbá a változások és egyéb fejlesztések visszavezetése sok esetben nincs megfelelően szabályozva, pedig világosnak kellene lennie, hogy milyen esetekben és milyen kódot lehet az eredeti projektbe visszajuttatni [213]. A nyílttá tett forráskód bizonyos jogi veszélyeket is hordozhat. Amennyiben a kód nem tisztázott állapotú szellemi terméket tartalmaz, a publikáló szervezetnek számolnia kell a pereskedés lehetőségével [30] ami sértheti a rendelkezésre állást.

Az integráció több szinten is megvalósulhat. Integrálhatunk pár soros kódot, vagy középszinten komponens osztályokat, metódusokat, vagy nagyobb skálán teljes fájlokat, funkciókat vagy komplett rendszert. [59] A

forrás birtokában az integráció sokkal rugalmasabbá válik és lényegében feloldja a bináris programkönyvtár interfészek mesterséges határait, viszont egyúttal el is veszítjük ezen interfészek nyújtotta biztonságot. A könyvtárak fejlesztői általában feltételezik, hogy ezeken a határokon belül szabadon módosíthatnak bármit, amire ha nem figyelünk oda kompatibilitási problémákat vagy hibákat vezethetünk be a rendszerünkbe.

A kontrollálatlan integráció súlyos veszélyforrás is lehet. A népszerű biztonsági segédszoftverek között igen sok a FLOSS termék (FS-H-P) amelynek felhasználása kapcsán ismert olyan eset, ahol a belső hálózatban sérülékeny nyílt forrású biztonsági könyvtárakat és keretrendszereket használnak fel a saját belső alkalmazásaik fejlesztésénél [207].

Összességében a komponensintegráció kapcsán négyféle kockázati tényezőt lehet megkülönböztetni:

1. komponens kiválasztás kockázata (nem a megfelelő vagy sérülékeny változatot építünk be);
2. komponens integrációjának kockázata (rosszul integráljuk a komponenst);
3. komponens karbantartásának kockázata (frissítések, új változatok hibákat vezetnek be, megszűnik a támogatás);
4. jogi kockázat[18]. [184]

A nagy számú FLOSS komponenst felhasználó cégeknek szükségük van valamilyen FLOSS menedzsment keretrendszerre, amely segítségével kontrollálni tudják a FLOSS felhasználásukat. A menedzsment keretrendszer követelményei az extenzív FLOSS-felhasználó cégek között végzett felmérés alapján az alábbiak[216]:

1. FLOSS komponensek követése és újrahasznosítása:
    1. Az eszköz azonosítsa a FLOSS komponenseket a kódbázisban;
    2. segítsen jelenteni a FLOSS komponenseket az architektúra modellben;
    3. frissítse a FLOSS komponenseket és metaadataikat;
    4. vezessen nyilvántartást a felhasznált FLOSS komponensekről;
    5. nyújtson lehetőséget a már egyszer felhasznált komponens újrahasznosítására.
2. Licenc megfelelőség:
    1. Az eszköz értelmezze a nyílt licenceket;

[18]E négy kategória utolsó elemével a szabályozást tárgyaló fejezetben foglalkozom (FS-Sz).

2. dokumentálja a komponens licencét a szervezet adattárában;
3. segítsen keresni az adattárban;
4. képes legyen igazolni, hogy a komponens licence megfelel a projekt irányának;
5. segítsen olyan licenc kiválasztásában ami kompatibilis az integrál komponensek licencével.

3. FLOSS komponensek keresése és kiválasztása:
    1. Az eszköz nyújtson lehetőséget a FLOSS komponensek közötti keresésre;
    2. segítsen megtalálni a legjobb komponenst;
    3. legyen képes megbecsülni a komponens használatának költségét.

4. Egyéb követelmények:
    1. Az eszköz képes legyen megtalálni és megelőzni a komponensben található sérülékenységeket;
    2. dokumentálja és kommunikálja a cég FLOSS kezelési stratégiáját, rendszabályait és javasolt gyakorlatát;
    3. segítsen a felhasználónak megismerni a FLOSS kezelés és megfelelőség elveit a fejlesztés és a közösséggel való kommunikáció során.

Összefoglalva, a FLOSS komponens felhasználás biztonsági problémákat okozhat, menedzsmentje kihívásokkal teli komplex feladat. Különösen fontos a bizalom és megbízhatóság, a komponens biztonsági modellezés kérdése és a tesztelés [217]. A közösséggel való kommunikáció nehezen elkerülhető és nem is javasolt. A változásokat lehetőség szerint vissza kell vezetni a forrásprojektbe, ugyanakkor bizonyos információk nem oszthatók meg a nyílt forrású közösséggel. Külön figyelmet kell rá fordítani, hogy ezek az információk ne szivároghassanak ki [218].

A téma összetettsége folytán komoly igény mutatkozik a FLOSS komponens integráció célzott oktatására, hiszen a fejlesztőknek egyre inkább oda kell figyelniük rá, hogyan tartják a kapcsolatot a FLOSS világával. [97]

2.3.3 Gyártófüggetlenség (FS-H-Gy)

A FLOSS egyik csábító előnye, hogy felhasználásával csökkenthető, vagy akár meg is szüntethető a vendor-lock-in jelensége, azaz a szállítóktól, gyártóktól való függés. Nyílt környezetben nem szükséges elkötelez-

ni magunkat egyetlen cég vagy környezet mellett sem. Továbbá a FLOSS nem kényszeríti a szervezetet szoftver és hardverspirálba, ugyanis a régebbi hardveren is jól használható és a frissítéseket sem követeli meg [219]. Bár a biztonsági frissítések nélküli szoftverhasználat komoly veszélyt jelenthet, bizonyos nyílt projektek (mint minden nagyobb Linux disztribúció) szabott ideig tisztán biztonsági frissítéseket is garantálnak, további képesség-fejlesztések kikényszerítése nélkül.

A rendelkezésre álló felmérések [168, 220, 221] azt mutatták, hogy a közszolgálati szektorban a gyártóktól való függetlenedés a FLOSS megoldások egyik legfontosabb előnye. A függetlenség két kulcsfontosságú pontja, hogy a szervezet ne függjön külső szereplőktől az adatbiztonság terén és ne kötődjön bizonyos szoftver termékekhez vagy gyártókhoz [12]. A kívánt függetlenség elérhető tisztán FLOSS rendszerekkel, vagy üzleti és nyílt rendszerek kombinálásával illetve nyílt szabványok használatával.

A FLOSS használatával mérhető módon csökkenthető a függőség [204]. A függetlenség növekedésével a szervezet nyomást tud gyakorolni a beszállítókra [162].

Bár a függetlenség közvetett módon javítja a biztonságot, hiszen nagyobb rugalmasságot szélesebb mozgásteret biztosít, közvetlen biztonsági hatást nem azonosítottam.

2.3.4 Felhasználói tábor (FS-H-F)

A FLOSS felhasználók eloszlása érdekes mintát mutat. Elterjedt nézet, hogy a nyílt forrású rendszereket programozók fejlesztik programozóknak vagy éppen saját maguknak. Ez persze erős általánosítás, de a feldolgozott kutatások azt mutatják, hogy bizonyos mértékig valóban létezik ilyen jelenség.

A személyi felhasználók közt többségben vannak azok, akik “programokat írnak” még ha nem is feltétlenül professzionális szinten, hanem saját céljaik elérése érdekében ami általában valami más, könyvelés, statisztika, webtervezés, kutatás vagy szórakoztatás [222]. A Stack-Overflow kommunikációjának analízise azt mutatta, hogy a fejlesztők tudatában vannak a FLOSS nyújtotta előnyöknek és könnyebben állnak át az ilyen megoldásokra [161].

A fiatalabb korosztály tagjai statisztikailag hajlamosabbak FLOSS rendszerek használatára, jobban ismerik azt [160] ami valószínűsíti, hogy a FLOSS felhasználószám növelését a fiatalabb rétegen keresztül lehet

megcélozni. [132] Ezzel szemben az ellenzők általában tapasztaltabbak, ugyanakkor kevesebb a kapcsolatuk a szabad szoftverekkel.

Nem meglepő módon a szabad szoftver közösségekben szerepet vállaló személyek körében nagyobb a FLOSS iránti igény, magasabbra is értékelik azt [223] és a felhasználótábor (részhalmaza) gyakran egyben maga a fejlesztő közösség is [72].

Hasonlóan magas a biztonsági szakemberek FLOSS felhasználása. A információ-biztonsági szakemberek általában megbíznak a FLOSS biztonsági eszközökben. Ugyanakkor érdekes módon a nagyvállalatok már kevésbé bíznak a FLOSS biztonsági eszközökben (amiben szerepet játszanak az alább kifejtett okok) [224]. Li eredményei azt mutatták, hogy a felhasználók biztonság-kritikus területeken mindenképpen igen óvatosak, akár COTS[19] akár FLOSS termékről van szó, a szoftverek biztonsági aspektusát így nem igazán lehet megkülönböztetni. [70]

A szervezeti felhasználók közt a magukat innovatívnak tartók hajlamosabbak kihasználni a FLOSS nyújtotta lehetőségeket és nagyobb eséllyel jelölik meg jövőbeli célként [78]. A FLOSS-használó szervezetek általában kicsik és kevésbé kritikus IT részleggel rendelkeznek mint a FLOSS-t nem vagy kisebb mértékben használó vállalkozások. Ez bizonyos értelemben ellentmond annak nézetnek, hogy az IT szempontból kritikusabb, nagyobb vállalkozások fogékonyabbak az innovációra. A jelenség oka az lehet, hogy a kis vállalkozások az alacsony költség miatt ugródeszkaként használhatják a FLOSS rendszereket, hiszen így gyorsan (és olcsón) hozzájuthatnak a legfrissebb technológiákhoz. Ugyanakkor a támogatás és a szolgáltatás körüli bizonytalanságok miatt a nagyobb vállalatok felelős vezetői inkább kockázatkerülő magatartást követnek [225]. A mikro és kisvállalkozások a fejlődőfélben lévő FLOSS leggyakoribb támogatói. Gyakran járulnak hozzá a projektekhez, ugyanakkor a méretük miatt ők tudnak a legkevesebbet gazdaságilag profitálni a szabad szoftverekből, számukra tehát elsősorban a korábban említett “gyors-rajt” effektus biztosítja az előnyt. A középvállalkozások kisebb mértékben de hajlandóak kísérletezni fejlődőfélben lévő szabad szoftverekkel, sőt bizonyos esetekben saját szaktudásukat vissza is forgatják a közösség javát szolgálva. A nagyvállalatok általában biztosra mennek és kizárólag a bejáratott, érett megoldásokat hajlandóak felhasználni, amelyek jelentős ideje vannak a piacon és amelyek esetén a biztonsági kockázat a lehető legkisebb. A nagyvállalatok tehát elsősorban erőforrás optimalizálás céljából veszik igénybe a FLOSS termékeket. [226]

[19] Commercial Off-The-Self : dobozos termékek, nem helyi, egyedi megoldások.

Érdekes módon Spinellis statisztikai elemzése azt mutatta, hogy a gyorsan fejlődő vállaltok és a különösen termelékeny munkavállalók egyáltalán nem részesítik előnyben a FLOSS rendszereket, sőt, ennek épp ellenkezője igaz [7]. Az eredmény csak első látásra meglepő, hiszen a szabad szoftver erősségének számító testreszabhatóság (lásd alább) nem nyújt valódi előnyt, ha a piaci termékek – igaz magasabb árért – gyorsabban és hatékonyabban le tudják fedni az összes igényt. Ezt pedig mindig megtehetik akár a FLOSS termék vagy annak komponenseinek felhasználásával is. A szabad szoftvereknek tehát továbbra is az alacsony költség marad mint versenyelőny.

2.3.5 Jellemző piaci szegmensek (FS-H-P)

A FLOSS termékek nem egyformán vannak jelen az egyes piaci területeken. A FLOSS jelenség jellemzően erős az IT szektorban. Jól ismert zászlóshajókat találhatunk itt az operációs rendszerek, webkiszolgálók, böngészők, szoftverfejlesztés, verziókezelés és adatbázis-kezelők piacán. [161] Hasonlóképpen erős a jelenléte a middleware [214], bioinformatika, katonai számítások és az akadémiai kutatások terén [227] sőt bizonyos területeket akár teljesen uralhat is [228, 229]. Ezek közül a felhőtechnológia különösen figyelemre méltó, hiszen a FLOSS potenciális problémái a felette futó rendszer sértetlenségét is veszélyeztetik[230].

Érdekes módon a magában is gyakran Létfontosságú Rendszerelemnek számító közszféra információs rendszereiben növekvő érdeklődés tapasztalható a FLOSS rendszerek iránt. Míg korábban elsősorban a FLOSS webszolgáltatásokat használták a szabványkövetés, módosíthatóság és függetlenség nagy vonzerőt jelentenek ebben a szektorban. [12, 221] Ebben a szektorban a polgárok jogai odáig is terjedhetnek, hogy az adatok előállításának pontos módszereit is megismerhessék, ami a forrás nyíltsága nélkül aggályos [12]. A FLOSS jelentősége egyértelműen növekszik például az Egyesült Királyság közszolgálati szektorában, a politika egyre inkább felismeri az előnyös hatásokat [12]. Belgiumban Viseur erős jelenlétet azonosított a voip, erp és a képzés terén is[231]. A globális gazdasági hatás előnyeivel a vonatkozó részben foglalkozom (FS-G-G).

Más területeken ugyanakkor nincs vagy csak elvétve található jó minőségű (vagy akár minimálisan versenyképes) megoldás. Nincs értékelhető jelenlét például az ipari célszoftverek piacán [220]. Jellemző a FLOSS használat az ad-hoc biztonsági megoldások kialakításakor és a tesztelések során, jelentős szere-

pet játszik az esemény utáni szakértői elemzésekben[20], ugyanakkor a kritikus rendszerekhez már szinte minden esetben üzleti alkalmazásokat használnak a nagyvállalati szektorban [172].

Bizonyos biztonságkritikus területek kifejezetten kerülik a nyíltsággal járó kockázatokat. Ilyen a DRM[21] védelem, az ATM és POS terminálok vagy a smart card ipar területei[232].

2.3.6 Adatmigráció (FS-H-A)

A FLOSS változatra való áttérés általában nem pusztán egy új szoftver telepítését jelenti. A meglévő adatokat migrálni kell, a rendszer más komponensekre irányuló interfészeit újra kell írni. Ez különösen nehézkes lehet a FLOSS diszkriminatív tulajdonsága miatt (FS-Sz-L). [108]

Egy esetleges migráció során szervezeti problémákkal is szembe kell nézni. A vállalati vezetők a változással szembeni ellenállást, a házon belüli FLOSS szakértelem hiányát és a FLOSS bevezetésével kapcsolatos tapasztalatok hiányát jelölték meg fő akadályként. [108]

A nagyvállalatok komplex információs rendszereiben a két legjelentősebb egyedi érték az idők során létrehozott kifinomult algoritmusok és az azt karbantartó kompetens csapat. Az algoritmusok és eljárások általában igen nagy mennyiségű kód integráns részét képezik, amelynek változtatása meglehetősen nehéz. A hosszú idő alatt felépített csapat és az összegyűjtött know-how csak lassú változásokra képes. Nem meglepő, hogy a felső vezetés a rendszer két legfontosabb értékét nem szívesen kockáztatja, ha úgy érzi a FLOSS bevezetése veszélyeztetné őket. [167]

Tekintve, hogy a migrációs folyamat nem mindig visszafordítható, a migrálás az integritást sértő esetleges migrációs hibáktól eltekintve is sértheti a rendelkezésre állást elvét, tekintve, hogy a migrálás előtti rendszermentések az újr rendszerbe csak komoly nehézségek árán tölthetők vissza.

2.3.7 Egyedi igényekhez alakíthatóság, testreszabás (FS-H-T)

A szoftvergyártó-felhasználó értékesítési csatornában a legtöbb esetben megjelennek köztes szereplők, az úgynevezett rendszerintegrátorok. A gyártó nem közvetlenül értékesíti termékét, hanem a rendszerinteg-

[20] Angol nevén “Forensic Analysis”
[21] Digital Right Management

rátoron keresztül, melynek termékében központi komponens az adott szoftver, de kiegészítő termékeket és szolgáltatásokat kínál hozzá. A jó minőségű szoftverrendszer előállításánál a FLOSS szinte teljes mértékű módosíthatósága nagy előnyt jelent, hiszen a forráskód birtokában elvben bármilyen átalakítás elvégezhető. [233] Egyes vásárlóknak olyan speciális igényei lehetnek, amelyeket a szabványos változattal nem lehet kielégíteni, a kód változtatásának lehetősége pedig megoldást jelenthet erre a problémára, ami az integrátorok számára különösen előnyös. [30] Az fejlesztők számára egyetlen késztermék átalakítása kisebb befektetéssel jár hasonló eredmények mellett, mint egy teljesen új termék létrehozása. Teljes funkciókhoz jutnak így ingyenesen, a hiányzó képességeket már könnyebb pótolni. [234] A lokalizáció, azaz a helyi nyelvhez, szokásokhoz illesztés is könnyebben elvégezhető ha a forrás és a fordítást végző közösség rendelkezésre áll. Ez a képesség a FLOSS rendszerek egyik erőssége [108] bár a lokalizációt célzó SaaS szolgáltatások[22] részlegesen csökkenthetik az üzleti szoftverrendszer lokalizációs terheit, az azonnali és egyedi módosítás továbbra is csak a forrás birtokában lehetséges.

A nyílt kód segítségével egyúttal magasabb biztonsági szint is elérhető, hiszen az esetleges hibák és biztonsági rések kijavíthatóak [219]. A változtatásokat saját időbeosztás szerint bármikor el lehet végezni, nem kell várakozni a gyártó kiadási ciklusaira [60]. A zárt forrású megoldásokba nem lehet belenyúlni, vészhelyzetben nincs lehetőség a reakcióra, az egyetlen azonnali megoldás a folyamat ismételt újraindítása ami nem feltétlenül vezet eredményre. Nyílt forrás esetén megfelelő szaktudás mellett legalább a lehetőség adott [40].

A felhasználók szemszögéből nézve a testreszabhatóság kérdése árnyaltabb. Az üzleti megoldások általában rendelkeznek valamilyen szintű beállítási lehetőséggel, de a FLOSS rendszerek testreszabási lehetőségei messze meghaladják ezeket [92]. Problémát jelent viszont, hogy a testreszabáshoz általában jelentős szakismeret és idő szükséges, ami egy átlagfelhasználó esetén sokszor túlzó elvárás. Ezt igazolták Spinellis korábban említett eredményei, amelyből látható, hogy a gyorsan fejlődő, produktív felhasználók és cégek inkább a piac által számukra előre testreszabott megoldásokat részesítik előnyben. [7]

A változtathatóság előnyeinek pontos értékét azonban nem könnyű felbecsülni [60] a változtatásnak pedig adminisztrációs ára van. Az új verziók változásait a szervezetnek követni kell, ha nem akar lemondani a biztonsági frissítések nyújtotta előnyökről, és ha nem vezeti vagy nem tudja visszavezetni a változtatásait, a közösség támogatása jelentősen csökkenhet ha később problémák merülnek fel. A fentiek miatt a

[22]ilyen SaaS szolgáltatást nyújt például a Transifex (https://www.transifex.com/)

változtatás kizárólag indokolt esetben javasolt. [181, 235]

Összefoglalva, a testreszabhatóság elsősorban az integrátorok és fejlesztők számára nyújt valódi előnyöket, a végfelhasználó csak akkor tudja kiaknázni ezeket a lehetőségeket, ha magasan képzett és megfelelő mennyiségű idő áll rendelkezésére. Amennyiben a szervezet belső erőforrásból végez rendszerfejlesztéseket a testreszabhatóság jelenthet biztonsági előnyt, egyéb esetben ez a hatás nem releváns.

2.3.8 Összefoglalás

A FLOSS felhasználói szempontból bizonyos mértékig azonos tulajdonsággal rendelkezik mint a COTS termékek (az USA szabályzata egyenesen COTS terméknek tekint minden FLOSS terméket), azonban van néhány fontos eltérés. A számtalan változat miatt a számunkra megfelelő termék kiválasztása nem egyértelmű, a megszokott minősítő rendszerek nem használhatóak, egyedi adatbázisok és modellek szükségesek.

Az igen széles körű FLOSS komponens integráció sajátos problémákkal küzd, amelyből a legfontosabb a helyi változtatások kezelésének és a White-box integráció során elvégzett ellenőrzés meglétének kérdése.

A gyártófüggetlenség vonzó előny, de közvetlen biztonsági hatást nem sikerült azonosítani. A FLOSS felhasználói tábora bizonyos eltérést mutat az üzleti termékek felhasználói táborától, azonban ez az eltérés nem szignifikáns. Míg bizonyos piaci szegmensekben a FLOSS piacvezető, más szegmensekben rendkívül alacsony a jelenlét és a választék.

A FLOSS migráció szempontjából nem különbözik jelentősen az üzleti termékektől. A vezetés gyakran ad hangot aggodalmának, de ezek az aggodalmakat inkább a bizonytalanság generálja, technikai szempontból nincs jelentős különbség a migráció terén.

2.7. táblázat: A H kategóriában azonosított problémák

kód	szint	leírás	sajátosság
SH01	1	A szervezet nem megfelelő minősítő rendszert alkalmaz a FLOSS termék minősítésére.	FS-H-M
SH02	1	Megfelelő változat kiválasztása nehéz és időigényes. Emiatt előfordulhat, hogy nem a megfelelő változat kerül alkalmazásra.	FS-H-M

kód	szint	leírás	sajátosság
SH03	2	A kontroll nélküli (esetleg rejtett) integráció a hibás komponensek révén sérülékenységek akaratlan beépítéséhez vezet.	FS-H-I
SH04	1	A FLOSS komponensek minősítésére nem alkalmas a COTS esetén megszokott módszertan.	FS-H-M
SH05	2	Az integrátor nem ismeri megfelelő mélységben az integrálandó komponenst így hibákat vezet be az integráció során.	FS-H-M
SH06	3	Az integrált komponensen végzett módosítások összeütközésbe kerülnek az eredeti kódbázissal, ami a frissítések során hibákhoz vagy sérülékenységhez vezethet.	FS-H-T
SH07	4	A szükséges módosításokat nem sikerül visszavezetni, a közösség elutasítja így a kódbázis ütközéseit folyamatosan figyelni kell, ami időigényes és hibalehetőségekkel terhelt.	FS-H-M
SH08	1	Az integrált komponens összetett függőségein keresztül hibás (és ellenőrizetlen) alkomponensek szivároghatnak a fejlesztésbe.	FS-H-M
SH09	4	A módosítások visszavezetése során érzékeny információ szivárog ki az upstream projektbe.	FS-H-M
SH10	1	Kompatibilitási problémák miatt a szervezet által alkalmazott felügyeleti rendszer nem támogatja a FLOSS változatot.	-

2.8. táblázat: A H kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JH01	1	Formális tanúsítás híján a felhasznált komponensek kódját megbízhatósági modellekkel kell vizsgálni (pl. CodeTrust).	SF05, SH01, SH02
JH02	3	A kód szabad módosíthatósága miatt várakozás nélkül javíthatóak a sérülékenységek, amennyiben a kód módosításához szükséges erőforrás és szakismeret rendelkezésre áll.	
JH04	2	A felhasználandó FLOSS komponenseket erre szakosodott szakértők révén szerezzük be.	SF05, SH04, SH05, SH08, SJ01, SM06
JH05	1	A FLOSS komponens kiválasztásának folyamatát és körülményeit dokumentálni kell.	SF05, SF10, SH05
JH06	4	A kód változásait lehetőség szerint vissza kell vezetni az eredeti projektbe.	SF07, SF11, SH06
JH07	3	A fejlesztést végző részlegnek legyen megfelelő verzió-követési terve.	SF07, SF10, SF13, SH06

kód	szint	leírás	probléma
JH08	3	A beszállító vagy a fejlesztést végző részleg alkalmazzon FLOSS komponens kezelő keretrendszert.	SF07, SF10, SF13, SH03, SH06, SH08, SS02, SS03
JH10	4	A forráskódban minden nem publikus információ világosan definiált legyen. A változások visszavezetésekor automatizált rendszerrel meg kell akadályozni a minősített információ kiszivárgását.	SF07, SH09
JH11	2	Képzésekkel kell tudatosítani a komponens integrációval járó veszélyeket és a biztonságos felhasználáshoz szükséges eljárásokat	SF13, SH04, SH05, SH06, SH08, SH09

2.4 FLOSS mint termék, terméktulajdonságok (FS-J)

A FLOSS előnyeivel kapcsolatos kérdésekre adott válaszaikban a megkérdezettek gyakran említik a megbízhatóságot, biztonságot, minőséget, teljesítményt, rugalmasságot, alacsony költséget, jogi rugalmasságot, a gyártófüggetlenséget, megnövekedett együttműködési lehetőségeket [102, 236–238]. Hátrányként általában a központosított támogatás hiányát, kompatibilitási és telepítési problémákat, felhasználó-barátsággal kapcsolatos aggályokat, irányítás lehetőségének hiányát, változással szembeni ellenállást, a nem megfelelő marketinget és a magasabb betanítási költséget szokás említeni [236, 239, 240].

A 2.9-2.12. táblázatok rendszerezett formában mutatják be milyen előnyöket és hátrányokat érzékelnek a felhasználók a FLOSS termékkel kapcsolatban.

2.9. táblázat: Műszaki előnyök Morgan és Finnegan szerint [236]

Megbízhatóság	gyakran felhozott előny, különösen magas rendelkezésre állás szükségessége esetén
Biztonság	első sorban a forrás nyíltsága miatt, bár ez a hatás inkább csak az idősebb, érett projektekre jellemző
Minőség	forrása a magasabb fokú ellenőrzés, a fejlesztés és tesztelésminősége
Teljesítmény	elsősorban tárolás és sebesség terén, ugyanakkor gyakran a fordítottját állítják

2.9. táblázat: Műszaki előnyök Morgan és Finnegan szerint [236]

Rugalmasság	változtatások lehetősége, testreszabás, kísérletezés és választás szabadsága
Kompatibilitás	időtálló formátumok használata (gyakran vitatott)
Harmonizáció	jobb harmonizációs lehetőségek az együttműködés terén

2.10. táblázat: FLOSS üzleti előnyei Morgan és Finnegan szerint [236]

Alacsony költség	kisebb díjak licenszelés, frissítés, vírusvédelem terén, alacsonyabb teljes költség.
Jogi rugalmasság	elsősorban költségcsökkentő hatása miatt fontos
Gyártófüggetlenség	szabad választást tesz lehetővé, irányítás érzetét kelti, de FLOSS esetén is elképzelhető!
Innováció ösztönzése	a forrás elérhetősége segíti az innovációt és új lehetőségeket nyújt
Üzleti lehetőségek	lehetőséget ad kisebb csapatmérettel dolgozni, ami növeli a termelékenységet

2.11. táblázat: FLOSS műszaki hátrányai Morgan és Finnegan szerint [236]

Kompatibilitási problémák	meglévő technológiákhoz,tudáshoz és feladatokhoz illesztés nehézkes
Szakértelem hiánya	a hiányzó szakértelem a tájékoztatás hiányából fakadhat
Gyenge dokumentáció	a dokumentáció elavult, frissítése teljesen leállhat a fejlesztés során
Interfész bizonytalanság	nem egyértelmű hogy az egyes kiadások melyiket használják
Funkcionalitás	az integráció és funkcionalitás nem éri el az üzleti termékekét
Roadmap hiánya	a szervezet nehezen látja át a stratégiai irányt, ez sokszor teljesen hiányzik is

2.12. táblázat: FLOSS gazdasági hátrányai Morgan és Finnegan szerint [236]

Támogatás hiánya	nincs biztonságérzet a támogatás és a háttércég hiánya miatt
Tulajdonos hiánya	nincs felelőségre vonható személy vagy szervezet

2.12. táblázat: FLOSS gazdasági hátrányai Morgan és Finnegan szerint [236]

Forrás elérhetősége	van, hogy kényelmetlenséget okoz a forrás elérhetősége (rosszul informáltság?)
Marketing hiánya	tulajdonos híján nincs marketing, nincsenek marketing források, "szájhagyomány útján" terjed
Képzés költsége	képzés költsége magasabb lehet, a képzés minősége viszont általában jobb
Kompetencia felkutatása	nehéz a kompetens fejlesztőket és szakértőket megtalálni

Ezek egy részével más fejezetekben foglalkozom (FS-Sz; FS-H-Gy; FS-H-T; FS-H-I; FS-H-Gy; FS-T-S; FS-Sz-M) a fennmaradó tulajdonságok hatását ebben a fejezetben elemzem.

2.4.1 Technikai átláthatóság (FS-J-Á)

A FLOSS termékek biztonsági szempontból egyik leghasznosabb tulajdonsága a forrás szintű átláthatóság. Ebből az átláthatóságból számos potenciális előny származik, amelyeket az alábbiakban mutatok be.

Egy zárt rendszer stabilitásáról igen nehéz adatokat gyűjteni, hiszen a vállalkozások a vonatkozó adatokat stratégiai információként kezelik amelynek kiszivárgása súlyos üzleti hátrányokhoz vezethet. Ezzel szemben FLOSS esetében viszonylag egyszerű adatokat gyűjteni amivel előrejelző modellek kalibrálhatók és validálhatók. [118]

Közszolgálati rendszerekben követelmény lehet az adatelőállítás teljes folyamatának publikálása, hogy az állampolgárok nyomon tudják követni az információ létrehozásához használt eljárásokat és módszereket. Ehhez a felhasznált szoftvernek jól dokumentáltnak kell lennie és nyílt minősítési folyamaton kell keresztülmennie. A FLOSS jelentős versenyelőnnyel rendelkezik ezen a téren [12].

Az átláthatósághoz kötött magasabb biztonság kérdése általában vitatott. Nagyon nehéz eldönteni, hogy a zárt vagy nyílt verziók a sérülékenyebbek. Mind nyílt mind zárt szoftverekben voltak biztonsági sérülékenységek, amelyek éveken keresztül felismeretlenek maradtak [207]. Bizonyos problémák jól orvosolhatóak a forrás ismeretében. A komponensek megbízhatóságának minősítése és javítása megoldható például külső (harmadik) fél felkérésével, amely a lehetséges biztonsági hiányosságokat feltárja [61].

A szoftver belső szerkezeti felépítése gyakran ismeretlen. Előfordulhat, hogy a szerkezet feltárásának egyetlen lehetséges módja a forrás vizsgálata [241]. Hasonlóképp, a forrás alapján a rendszer belső minősége is felmérhető, amit általában lehetetlen közvetlenül kivizsgálni zárt forrás esetén [179]. A belső minőségellenőrzése automatizálható, a szakirodalomban számos metrika létrehozására találunk javaslatokat (lásd. McCabe Cyclomatic Complexity, Chidamber and Kemerer objektum orientált metrikái, Halstead complexity metrika). FLOSS használata esetén lehetőség van tehát olyan hiba és sérülékenység indikátorok előállítására amelyek alapján minőségi és biztonsági előrejelzések készíthetők [242]. Ugyanakkor az gyártók álláspontja szerint az üzleti megoldások elemzése is vezethet ellenőrizhető eredményekre, amennyiben a gyártó részletes dokumentációt tesz közzé és megfelelő tesztek állnak rendelkezésre a szoftver helyes működését tanúsítandó. Ezek alapján a közösség vagy független szervezetek szintén ellenőrizhetik az minőséget. [243] Pontosan ezen minősítési hiányosság folytán tartalmaznak a biztonsági keretrendszerek (pl. NIST) tesztelési kritériumokat, és végső soron ezt a hiányosságot orvosolja bizonyos szempontból a Common Criteria minősítési rendszere. Az egyedi, "házon belüli" minősítés elméletben ugyan jól hangzik, ugyanakkor az ehhez szükséges jelentős erőforrásigényből kifolyólag csak ritkán kivitelezhető a gyakorlatban. Ha ilyen minősítésre kerül sor, a közösség érdekében mindenképpen érdemes azt publikálni, egyéb esetben az ellenőrzési folyamat erősen redundánssá válik. Természetesen ez esetben nyitott marad a minősítés hitelességének kérdése.

2.4.2 Felhasználói dokumentáció (FS-J-D)

Szervezett támogatás híján FLOSS alkalmazások esetén különösen fontos a jó minőségű dokumentáció elérhetősége, hiszen gyakorta e dokumentációnak kell betöltenie a támogatás szerepét is. Ennek megfelelően a dokumentáció fontosságát a felhasználók magasra értékelik a felmérések során [209, 236–238]. A FLOSS projektek gyakran hoznak létre külön alprojekteket a dokumentáció gondozására [135], de a dokumentáció megvalósítása lehet automatizált vagy félautomatizált, ahol a forráskódba illesztett megjegyzésekből áll elő a dokumentáció (JavaDoc és hasonló meoldások).

Érdemes megkülönböztetni a felhasználói, fejlesztői és karbantartói dokumentációt. A fejlesztői dokumentáció kérdésével a fejlesztéssel foglalkozó fejezetben foglalkozom bővebben (FS-F-M). A felhasználói és karbantartói dokumentációt, különösen ha az önálló projektként fut, általában bárki bővítheti, ezért egyál-

talán nem ritka, hogy a dokumentáció szerzői nem programozók. A dokumentációt többnyire konzisztens licenszek védik (pl. Creative Common). [135]

Attól függően, hogy a projekt mennyire akkurátusan vezeti a dokumentációt, létezik-e a dokumentáció frissítésére irányú szabályozása, a dokumentáció minősége lehet naprakész, elavult vagy akár egyértelműen hibás is.

2.4.3 Használhatóság, hordozhatóság, funkcionalitás (FS-J-H)

A használhatóság direkt kapcsolatban áll az FLOSS felhasználási statisztikáival (ellentétben a megbízhatóság vagy a hordozhatóság kérdésével) azaz amennyiben a használhatóság és funkcionalitás nő, az adott FLOSS használata is növekszik. [92] Ugyanakkor a használhatóság hiánya gyakran szerepel a FLOSS rendszerek hátrányainak listáján. Ennek egyik oka a FLOSS közösségi kultúra egyedisége (FS-K-R; FS-K-Sz) illetve az átlagfelhasználó bevonásának hiánya. Egyes területeken, ahol a felhasználó és a fejlesztő közösség halmazának metszete jelentős létszámot képvisel a használhatóság kiváló szintet érhet el. Ilyen például a biztonsági segédszoftverek és a fejlesztőeszközök területe. Máshol, ahol nincs jelentős átfedés a két csoport közt, a végfelhasználó érdek-érvényesítő képessége igen csekély, ami végső soron erősen csökkenti az érzékelt használhatóságot. [83]

A kérdőívekben a használhatóságot magasra értékelő felhasználók általában előzetes FLOSS kapcsolatokkal rendelkeznek [78] vagy a használhatóságot az elérhetőséggel és a forrás nyíltságából eredő tulajdonságokhoz kötötték [162].

Nagyon gyakori, hogy a felhasználó úgy érzi, a FLOSS nem futáskész, a kezdeti beállítás időigényes, hiányzik az üzleti megoldásoknál megszokott, gyors, “varázsló alapú” indítási lehetőség [244]. Következésképpen még ha a tényleges funkcionalitás és használhatóság nem is marad el a versenytársakétól, az érzékelt használhatóság általában kisebb mértékű.

A varázsló-szerű beállítás előnye, hogy normál esetben józan biztonsági beállításokat tartalmaz, a felhasználó tehát különösebb időráfordítás nélkül elérhet egy kielégítő biztonsági szintet. Az időigényes kézi beállítás lehetséges veszélye, hogy a felhasználó türelmét vesztve nem megfelelő biztonsági szintet biztosító beállításokat alkalmaz. Személyes tapasztalataim is azt igazolják, hogy ez a gyakorlatban könnyen

előfordulhat.

Hordozhatóság alatt a felhasználók általában a szerényebb hardverigényen vagy szélesebb platformskálán való futás képességét értik [219]. Zárt forrás esetén a fejlesztők gyakran csak a végső futtatási környezet szűk halmazát ismerik, így nehezen garantálható, hogy a rendszer hordozható lesz az operációs rendszerek és platformok közt [60].

2.4.4 Interoperabilitás, kompatibilitás, szabványkövetés (FS-J-K)

A Létfontosságú Információs Rendszerelemek nagy mennyiségű kritikus adat megőrzéséért felelősek. Az információ hosszú távú elérhetősége azonban nem biztosított, ha a rögzített információ olyan formátumban kerül tárolásra amely a szoftverek új generációval már nem nyitható meg [233]. Hasonlóan fontos, hogy a komplex rendszer komponensei felismerjék egymás adatformátumait és ne legyen szükség az inkompatibilitásból adódó felesleges (esetleg információ vesztéssel járó) konverzióra. Ismert tény, hogy mindkét probléma hosszú távú orvoslásának kulcsa a nyílt szabványok használatában rejlik.

A FLOSS rendszereket általában nyílt szabványok terén erősnek, (a zárt forrású termékekkel való) kompatibilitás terén gyengébbnek szokás tekinteni. [168]

Di Penta megállapította, hogy az esetek legnagyobb százalékában (83%) a kormányzatban már eleve használt üzleti alkalmazásokkal való kompatibilitási problémák akadályozzák leginkább a FLOSS implementációját[78, 164]. Hasonló eredményre jutott Marsan az egészségügyi szektort vizsgálva. Az egészségügyi hálózat ugyanis jobbára zárt forrású IT megoldásokat használ, ami igen ritkán kompatibilis a FLOSS rendszerekkel technikai szinten[143]. Közismert példák az ilyesfajta inkompatibilitásra a Microsoft Office alkalmazásokkal való nehézkes együttműködés (az elvileg nyílt, XML alapú formátumok megjelenésének ellenére is), valamint a Linux operációs rendszer eszközmeghajtóinak hiánya bizonyos hardverekhez [204].

Az védett, üzleti formátumok használata a vendor-lock-in, azaz a gyártótól való függés kialakításának egyik bevált eszköze, amivel a gyártók szívesen élnek is, ha lehetőségük adódik rá. Megfordítva, a nyílt szabványok használatának megkövetelése általában nagyobb szabadságot biztosít a beszállítók kiválasztása terén [36]. Egyúttal a nyílt szabványok használata az előfeltétele a nemzetközi közös projektek zökkenőmentes lebonyolításának is [204].

A FLOSS támogatói gyakran érvelnek a szabad szoftverek nyílt szabványkövető tulajdonságával. A FLOSS termékek ugyan valóban szabványkövetőek, hiszen semmilyen okuk nincs rá, hogy ne legyenek azok [108] de ez a tulajdonságuk nem kizárólagos. Az az elvárás, hogy lehetőleg minden alkalmazás nyílt forrású legyen egyáltalán nem életszerű sőt, a szabványkövetés szempontjából nem is szükségszerű. Elegendő ha minden alkalmazás esetében megköveteljük a nyílt szabványok használatát, legyenek azok nyílt vagy zárt forrásúak [245].

Ugyanakkor kétségtelen, hogy a FLOSS használata de facto garantálja a nyílt formátumot (még ha a szabványosságot nem is feltétlenül) implicit módon pedig az interoperabilitás lehetőségét is, ami miatt használata előnyös lehet a közszolgálati szektor szolgáltatásai esetében [221].

Végső soron valószínűleg a nyílt forrás fejlődése és előretörése szerepet játszott abban, hogy ma már az üzleti alkalmazások jelentős része is szabványkövető, sőt konzorciumokon keresztül aktívan szerepet vállal a nyílt szabványok kialakításában.

2.4.5 Alacsony hibaszám, jobb kódminőség (FS-J-M)

A FLOSS egyik leggyakrabban emlegetett és hasonlóan gyakran vitatott tulajdonsága a hibák feltételezett alacsony száma. A feltételezés alapja az, hogy a forráskódot nyíltsága révén bárki megtekintheti, így az abban rejlő hibák előbb utóbb napvilágra kerülnek, a hibák felderítése és javítása gyorsabb mint a zárt forrás esetén. [51]

Az elmélet ellenzői szerint viszont abból, hogy valami bárki számár olvasható, még egyáltalán nem következik, hogy azt mindenki – vagy egyáltalán valaki – meg is nézi azt.

Sajnos a kérdés eldöntése egyáltalán nem egyszerű. A tények azt mutatják, hogy mind a nyílt és zárt forrású megoldások egyaránt tartalmaznak hibákat, ehhez elegendő néhány rövid keresést végezni a CVE[23] sérülékenység adatbázisban. A hibák számában látszólag nincs jelentős különbség, ami a józan ész alapján jelentheti azt is, hogy a felfedezetlen hibák száma is nagyjából azonos, de okoskodhatunk úgy is, hogy a forrás nyíltsága miatt a hibák felderítése könnyebb, így az adatbázisokban szereplő előfordulások az összes

[23] Common Vulnerability Enumeration, a MITRE vállalat nemzetközileg ismert sérülékenység adatbázisa amely egységes számozást biztosít a már felfedett biztonsági sérülékenységekhez.

hiba nagyobb részét fedik le, tehát a fennmaradó hibák száma kevesebb, következésképpen az összes hiba száma kevesebb.

Huynh és Miller az összes dokumentált sérülékenység döntő részét kiadó webalkalmazások több különböző sérülékenységét[24] célzó analízise során megállapította, hogy sérülékenységek számát tekintve nincs jelentős különbség a nyílt és zárt forrás között [246] bizonyos területeken a zárt más esetekben a nyílt szoftverek bizonyultak jobbnak, de a különbség egyetlen esetben sem haladta meg a 7%-ot. Achuthan indiai felmérése azt mutatta, hogy felmérésben résztvevők fele teljesen semleges álláspontot képvisel a nyílt és zárt forrású termékek biztonságát illetően [247]. Vouk kutatásai szerint a RedHat Fedora operációs rendszer biztonsági sérülékenységei relatív ritkák és mennyiségük időben állandó, a webalkalmazások elemzése során viszont néhány esetben növekvő, más esetben éppen csökkenő sérülékenység sűrűséget talált míg a sérülékenységek száma és eloszlása igen nagy eltéréseket mutatott [242].

Torzíthatja az eredményt az is, hogy a gyártóknak jó okuk van titkolni a sérülékenységeket. Egyetlen sérülékenység bejelentése átlagosan 0.63%-al csökkenti egy cég piaci értékét, a veszteség dollármilliókra rúghat. Kimutatható az is, hogy a piac nem bünteti kevésbé a cégeket ha maguk jelentik be a sérülékenységet mintha egy harmadik fél akad rá arra. Ugyanakkor a hibák felderítésének költsége a felderített hibák számának növekedésével párhuzamosan egyre nő, így a további hibák felderítésének költsége idővel meghaladja a várható megtérülést [71].

Az üzleti megközelítést és az ellenőrizhetőség problémáit jól illusztrálja a “Horus scenario” néven elhíresült sérülékenység csoport (CVE-2017-9851 – 9864). A sérülékenységeket feltáró holland szakember szerint a SMA által gyártott napelemek firmwarében fellelhető hibák segítségével széleskörű DDOS támadás indítható az európai elektromos hálózat ellen, amely jól szervezett támadás esetén kritikus méreteket ölthet [248]. A gyártó saját whitepaper kiadványában valamennyi CVE bejelentést tételesen cáfolja vagy bagatellizálja [249]. Mivel a termék nem nyílt forrású, a whitepaper lényegi információkat nem tartalmaz, a kérdés továbbra is eldöntetlen és nyitott marad.

Tovább nehezíti a kérdéskört, hogy a FLOSS termékek palettája igen diverz, egészen gyenge minőségű a csúcskategóriás projektekig terjed. Az elemzések általában a népszerű, gyakran használt változatokat célozzák, holott a kevésbé népszerűek közt igen gyakoriak a biztonsági sérülékenységek, alacsony teljesít-

[24] Az említett publikációban Huynh és Miller a következő sérülékenységeket vizsgálta: XSS, SQL injection, Code injection, Command Execution, Information disclosure, Privilege Escalation

mény, hibás és gyenge dokumentáció és holt kódok használata [234].

A kódminőség kérdése könnyebben elemezhető hiszen a kódminőségi mérőszámoknak köszönhetően kvantitatív módszerekkel ellenőrizhető. Aberdour 100 nyílt forráskódú alkalmazás elemzése során arra jutott, hogy a kódminőség meghaladja a várakozásokat és összemérhető az üzletileg elérhető alkalmazások minőségével [54]. Ennek egyik oka az lehet, hogy maga a modell kényszeríti a fejlesztőket átláthatóbb és szabványkövetőbb kód írására, részben a mindenki által látható munka presztízsértéke miatt, részben mert csak így garantálható a nagy létszámú, gyorsan változó és nemzetközi fejlesztői közösséggel való hatékony interakció. Az átlátható kód pedig végső soron a biztonsági ellenőrzést is megkönnyíti [250].

A nyílt forrás nagy előnye, hogy praktikusan lehetetlen elrejteni a kódminőségbeli hiányosságokat. Akár gyenge akár jó minőségű termékről van szó, a forráskód elemzésével az eredmény mindenki számára nyilvánvaló. A fórumok átnézése során további információ gyűjthető az esetleges sérülékenységekről, míg ezek az információk üzleti alkalmazások esetében legfeljebb csak apró mozaik-darabkákból állíthatók össze, a kód megtekintése pedig szinte mindig kötelezettség vállalással jár, amennyiben egyáltalán lehetséges. [251]

A fentiek alapján az alacsonyabb hibaszám és kódminőség hipotézisét nem tartom igazolhatónak. A vizsgált kutatások nem támasztja alá a kódminőség vagy hibák számának statisztikailag szignifikáns eltérését.

2.4.6 Összefoglalás

A FLOSS termékek belső tulajdonságaik tekintetében nem térnek el az üzleti verzióktól. A felhasználók gyakran említenek különféle problémákat ezek azonban elsősorban a támogatás hiányára, a nem megfelelő képzésre vezethetők vissza. A FLOSS támogatói hasonlóképpen hosszasan sorolják az előnyöket, ugyanakkor ezek az előnyök sem termékjellemzők, inkább a rugalmas felhasználást érintő előnyök. Az egyetlen valós terméktulajdonság a magasabb minőség és biztonsági szint, ez azonban nem ellenőrizhető a téma kapcsán folyamatosan ellentétes eredmények születnek. Mindkét modellből származnak kimagaslóan jó minőségű és gyenge termékek. A különbség elsősorban a minőség tanúsításának módjában és a tanúsítás ellenőrizhetőségében mutatkozik meg és nem a termék tulajdonságaiban.

Valós biztonsági hatást jelent a dokumentáció esetleges hiánya vagy gyenge volta, ugyanakkor ez sem specifikusan FLOSS jelenség, az üzleti termékek esetén is előfordul, a gyakoriság szignifikáns eltérése

pedig nehezen bizonyítható.

Gyakran említett eltérés a magas szintű konfigurálhatóságból származó nehéz beállítás és az ezt megkönnyítű automatizált “varázslók” hiánya. Ez az eltérés sem általános, ráadásul nehéz eldönteni, hogy az alapértelmezés megléte vagy hiánya hat-e hátrányosabban a biztonságra. Az ésszerű alapértelmezés segíthet elkerülni a kezdeti hibákat, ugyanakkor a dokumentáció figyelmen kívül hagyására ösztönöz, márpedi a beállítások részletes ismere nélkül nyilvánvalón alacsonyabb biztonsági szint érhető el, mint ha azok megértését a termék kikényszeríti.

A FLOSS termékek átlagosan hordozhatóbbak de a hordozhatóság kötvetlen biztonsági hatásait nem sikerült azonosítani.

Összességében megállapítottam, hogy a FLOSS terméktulajdonságai révén nem befolyásolja jelentősen a biztonságot.

2.13. táblázat: A J kategóriában azonosított problémák

kód	szint	leírás	sajátosság
SJ01	2	A dokumentáció és a tényleges képességek és szükséges konfiguráció közt jelentős eltérés lehet, ami biztonsági problémákhoz vezethet.	FS-J-D
SJ02	2	Megfelelő alapértelmezett beállítások híján a felhasználó (türelmetlenségből vagy figyelmetlenségből) hibás beállításokat alkalmazhat.	FS-J-H
SJ03	1	A meglévő zárt rendszerekkel való inkompatibilitás.	FS-J-K

2.14. táblázat: A J kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JJ01	1	Ki kell használni a nyílt forrás nyilvános adatait a termékminősítés kalibrációja és validálása során.	ST05
JJ02	1	A minősítés belsőleg is elvégezhető, bár rendkívül erőforrásigényes.	SF05, SM06, SM11
JJ03	1	Szükség esetén harmadik felet kell felkérni a biztonsági hiányosságok feltárására és a komponens minősítésére.	SF05, SM06, SM11

2.5 Közösség, szubkultúra (FS-K)

A nyílt modell munkaerejét adó közösség mind szerkezetében, mind motivációjában, vezetési struktúrájában gyökeresen eltér a zárt modell szociális struktúráitól. Az utóbbi időben elszaporodtak a nyílt modell módszereit utánzó kisebb startupok és a nagyobb IT vállalatok is átvettek elemeket a nyílt módszertanból, sőt aktívan támaszkodnak a nyílt modell közösségeire. A közösség képességei, szervezettsége jelentős hatást gyakorolhat végtermékre, annak biztonsági szintjére.

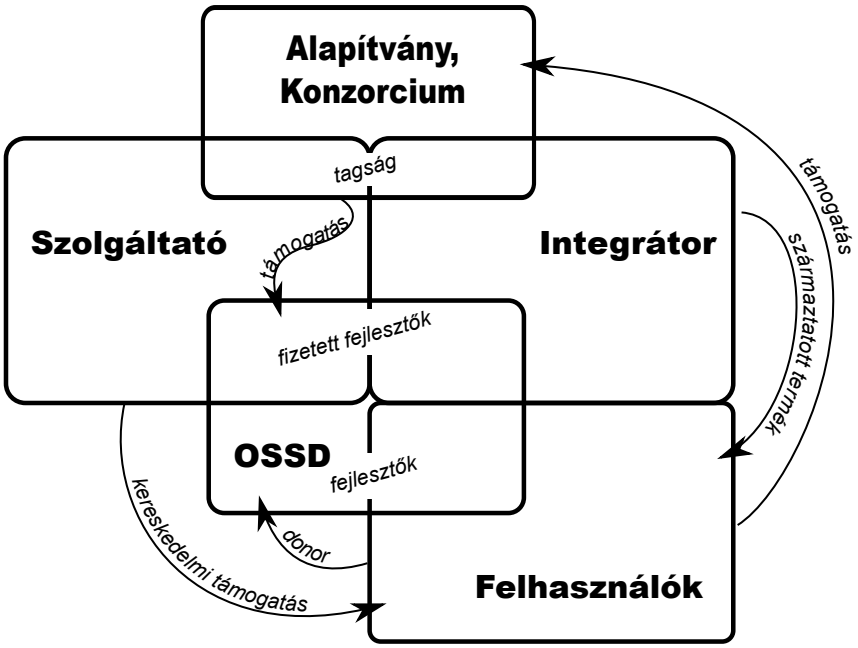
Emiatt – bár első pillantásra úgy tűnhet, hogy az információ és informatikai biztonsághoz nem sok köze van – fontosnak tartottam elemezni a klasszikus nyílt közösség felépítését és szociális jellegét, ugyanis a zárt modelltől való eltérések közvetve vagy közvetlenül hatással lehetnek a kockázatok beazonosítására és a problémák kezelésére is.

2.5.1 Szervezet (FS-K-Sz)

A klasszikus nyílt közösség önszerveződő, viszonylag gyorsan változó laza, moduláris hálózatot alkot, amelyben a résztvevők döntési mechanizmusai, szociális kapcsolatai eltérhetnek az üzleti modellben megszokottól.

Korábban nyílt közösség alatt magánszemélyek egy csoportját – elsősorban a fejlesztőket és tesztelőket – értették, mostanra azonban ez az elképzelés megváltozott. A gazdasági szereplők belépésével a nyílt közösség különféle csoportok komplex függőségi viszonyban álló halmazává vált, melyben jelentős szerepet töltenek be a terméket felhasználó integrátorok, a termékkel kapcsolatos szolgáltatásokat végző cégek és a jogi, társadalmi hátteret adó támogató szervezetek (alapítványok, konzorciumok) [252]. A nyílt közösséggel kapcsolatos csoportok szerepét és függőségeit a 2.5. ábra mutatja be.

Ezen csoportok együttes hatása alakítja ki a nyílt fejlesztési modell működési hátterét, jelentősen befolyásolva a végső termék használhatóságát és biztonsági szintjét. A FLOSS termékből közvetlenül hasznot húzó szolgáltatók és integrátorok fizetett fejlesztőkön keresztül igyekeznek biztosítani a projekt számukra kedvező irányát, míg a felhasználók – akik sok esetben egyben fejlesztők is – élvezhetik a két csoport nyújtotta termékeket és szolgáltatásokat.



2.5. ábra: OSSD szervezeti környezete (szerkesztette a szerző)

A támogatások és irányítás gyakran formális kereteket ölt konzorcium vagy alapítvány formájában, amelyet minden érdekelt fél támogathat, míg az alapítvány jogi és társadalmi hátteret biztosíthat a közös fejlesztésnek.

2.5.1.1 Fejlesztőközösség felépítése

Az OSSD közösségek sokkal inkább hasonlítanak egy szociális hálózatra [253] vagy a megoszló hálózatokra [254] mint hagyományos fejlesztőcsapatra. Bárki részt vehet bennük, bonyolult, határokon átívelő rendszert alkotnak[36, 107] és a kapcsolattartás döntő részben a hálózaton keresztül zajlik. A nagy földrajzi kiterjedés miatt az időeltolódás problémákat okozhat [255].

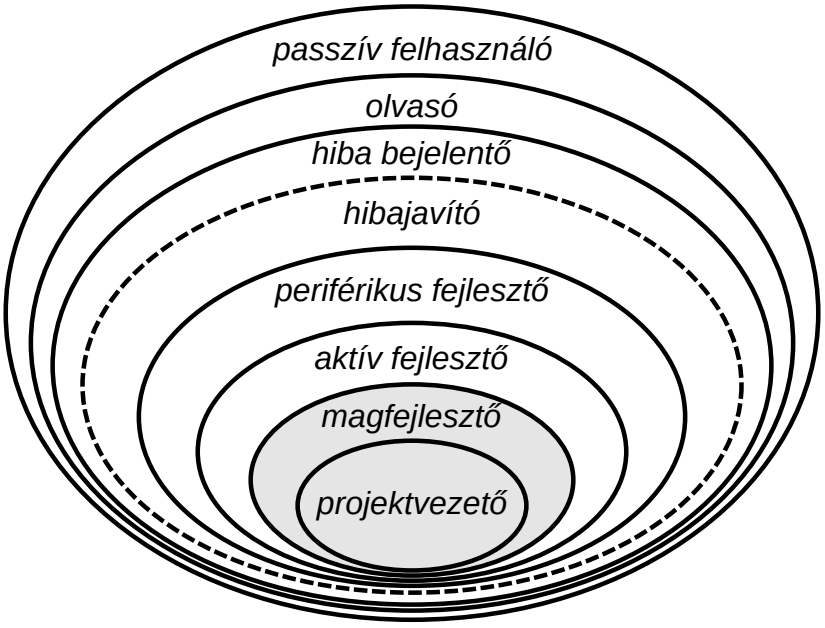
A közösség rugalmas, könnyen változik, szerkezetét tekintve pedig általában egy központi mag részből, és az azt körülvevő további külső rétegekből áll, hagymaszerű struktúrát hozva létre [256, 257] lásd fig. 2.6. A mag rész tagjai hozzák a legfontosabb döntéseket, általában csak nekik van joguk módosítani a központi forrástárakat és jobbára hosszú időn keresztül a projektben tartózkodnak[258]. A magot körülvevő fejlesztői holdudvar a magtól távolodva egyre ritkábban adományoz kódot és egyre kisebb szerepet vállal a fejlesztésből. A teljes fejlesztőtábor lehet egészen nagy, több ezer fős, de a közösség méretének növekedésével egyre kisebb a hozzáadott fejlesztőerő [259] igazolva, hogy a nyílt modellre is igaz a Ringelmann hatás.

A magfejlesztők kis csoportja adja a projekt gerincét, az érdemi munka döntő hányadát is gyakran ők végzik[107, 127], sőt a magfejlesztők nem egyszerűen csak többet dolgoznak, hanem minőségileg más munkát végeznek[260] (a hagyma "magja" tehát kissé eltérő). A mag mérete változhat de a legtöbb projektben 15 fő alatt marad [261]. A magfejlesztők között lehetnek "hősök", pótolhatatlan személyek ami termelékenységi szempontból hasznos [262] ám egyben kockázati tényező is, ugyanis a centrális emberek kiesése komolyan sértheti a csoport teljesítményét [263]. Az akadémiai projektként indult GIMP fejlesztése például több mint egy évre leállt, mert vezető fejlesztői befejezték az iskolát és elmentek dolgozni. Eddig tartott míg valaki más felvette a stafétabotot [123].

Az aktív fejlesztők jó rálátással rendelkeznek a projektre, idővel beléphetnek a magfejlesztők közé, bevonják őket a kulcsfontosságú döntésekbe, de szerepük marginális.

Az alkalmi fejlesztők csoportja a legnagyobb, ők egyetlen funkcióra vagy hibajavításra koncentrálnak. E csoport határán helyezkednek el az egyszeri fejlesztők akik valamikor hozzájárultak a projekthez, ám végül különféle okok miatt (időhiány, érdektelenség) végül eltávolodtak attól[264].

A nagyobb projektekben általában cégek is adományoznak kódot, illetve saját fejlesztőkön keresztül igyekeznek a mag közelébe kerülni [244]. Barham szerint a minőségbiztosítással foglalkozó közösségi tagok – ha vannak – külön réteget képeznek a hagyma-modellben és viszonylag elkülönülő kommunikációt folytatnak [265]



2.6. ábra: OSSD hagyma modell (saját változat Di Bella nyomán [256])

Bird szerint a magot körülvevő közösség felépítése sem homogén hanem kis csoportokra, modulokra bomlik, akik egymással sűrűbben kommunikálnak, esetleg saját központ központi tagjaik vannak[266] vagyis a homogén hagyma modell kissé megtévesztő. A szponzorált fejlesztők jelenléte tovább növeli a modularitást, az egy céghez tartozók között szorosabb a kapcsolat[267]. Ez a struktúra segít, hogy a fejlesztőszám növekedésével a kommunikációs overhead ne váljon kezelhetetlenné [268]. A decentralizált projektek jellemzően modulárisabbak mint a centralizáltak, a kisebbek pedig többnyire a hierarchikus típusból indulva érik el a decentralizált állapotot [269]. A viszonylag kis mag kialakulása nem hiányosság, hanem szükségszerű következmény, ugyanis egy nagyobb létszámú központi csoport tagjaira kezelhetetlenül nagy kommunikációs teher nehezedne.

Forrest lehetséges problémaként azonosította a hibabejelentők és a fejlesztők elkülönülését. A két csoport csoporton belül sokat kommunikál, ám a legjelentősebb fejlesztők a hibabejelentőkkel nem kommunikálnak [270], ami arra enged következtetni, hogy a gyors és hatékony hibajavítások érdekében a FLOSS felhasználó szervezetnek csatlakoznia kell a fejlesztéshez.

Bizonyos projektek saját biztonsági csapattal is rendelkeznek (pl. Drupal) akik a biztonsággal kapcsolatos figyelmeztetéseket figyelik és értékelik, illetve folyamatosan hibákat keresnek az alkalmazásban [271]. Ez a jelenség azonban nem általános és formális követelmények nélkül hatékonysága is megkérdőjelezhető.

Az irányított és szponzorált nyílt közösségek szignifikánsan több fizetett fejlesztőt alkalmaznak és jóval több a módosításra jogosult fejlesztők száma [272]. A magot körbevevő holdudvar végzi a tesztelés, hibakeresés nagy részét, a végső döntést és a kiadási ütemezést általában az irányító ipari szereplő határozza meg. Az irányított közösségekben érdemes megkülönböztetni a kettős licencelést alkalmazó, egyetlen cég köré épült projekteket[126], illetve a közös cél érdekében egyesülő cégcsoport által vezetett projekteket (technológiai konzorcium).

Liu szerint tisztán szervezetekből álló nyílt közösség is létrejöhet, ezt Community Source-nak nevezte [273].

2.5.1.2 Szociális struktúra

A közösség szociális struktúrája és ideológiai beállítottsága direkt hatást gyakorol a jelentkezésekre és a döntéshozatalra, így a teljes projekt teljesítményére [274]. Alapvetően szociális struktúráról lévén szó, a társakról alkotott vélemény és kapcsolattartás különösen fontos OSSD esetén. A felek gyakran csak egymás digitális valóját ismerik, gyakori a köszönetnyilvánítás, üdvözlés, a pozitív atmoszféra fenntartása. Az együttműködés alapja első sorban a bizalom és nem a tekintély. A bizalom elvesztésének fő oka a rossz kód (pl. kritikus hibák bevezetése, tervezési egység megsértése), kis mértékben szociális tényezők. A közösségekben régebb óta részt vevő felek sokkal hamarabb közös nevezőre tudnak jutni [275], vagyis a közösségi kommunikáció tanulható, fejleszthető. A személyes találkozások hatása egyértelműen pozitív [276] a találkozókon való részvétel tehát segít a kapcsolatépítésben, a prominens fejlesztők jelentős része gyakran személyesen is ismeri egymást [277].

2.5.1.3 Átláthatóság

Az közösség működése és felépítése szinte minden esetben átlátható. Igaz ez az információ nem közvetlenül publikált (nincs szervezeti diagram) de a metainformációk alapján könnyen felmérhető és elemezhető [274, 278]. Az OSSD projektek nyíltan működnek és minden érdeklődő közeledését szívesen veszik. Szabadon elérhető a szervezettel kapcsolatos (bár bizonyos területeken szegényes) minden dokumentáció, napló, beszélgetés [279].

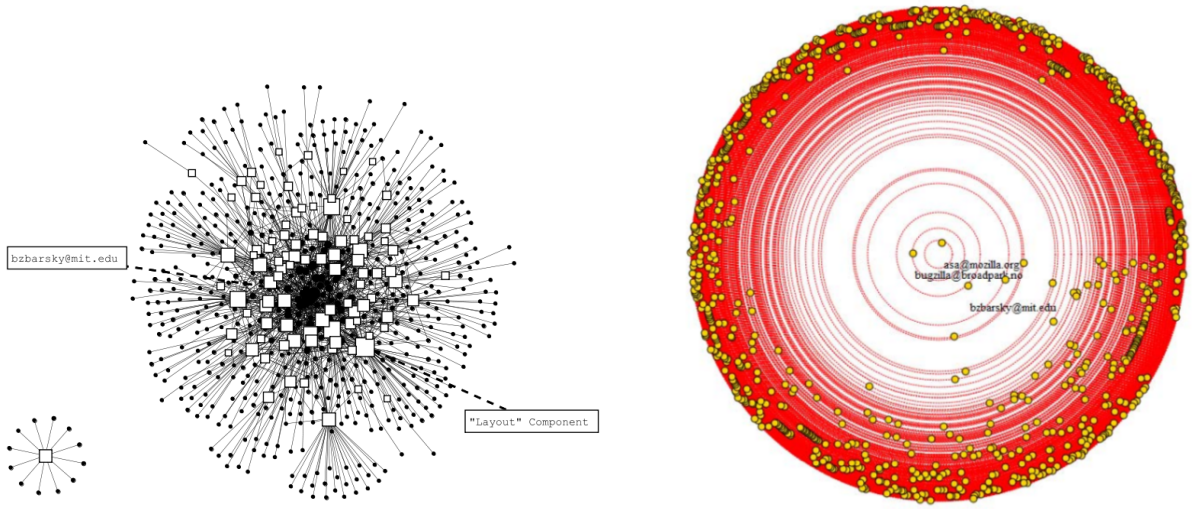
Az átláthatóság segíthet felmérni a nemzetbiztonsági kockázatokat annak ellenére, hogy a szervezet általában nemzetközi. Egy határon túli vállat üzletmenetébe meglehetősen problémás belelátni, nehezen átvilágítható a motivációja és azok a kényszerek amelyeknek – általában publikálatlanul – meg kell felelnie. Elég az elmúlt évek nagyobb adatgyűjtési botrányaira gondolni a Facebook vagy a Twitter kapcsán. Ráadásul értelemszerűen felmerül a kérdés, hogy vajon mekkora nyomást tud gyakorolni egy vállalatra a működési környezetét biztosító nemzetállam. A klasszikus nyílt közösségekre messzemenően nehezebb nyomást gyakorolni, pláne elrejteni annak nyomait.

A közösségre élénk kommunikáció jellemző és a kommunikációs adat szinte minden esetben könnyen hozzáférhető[280], igaz az utóbbi években a kommunikációs csatornák portálokra tolódásával (pl. Slack) ez

a nyíltság – legalábbis az információ birtoklása és perzisztenciája tekintetében – veszélybe kerülhet[33, 281]. Általában megállapítható az egyes szereplők munkaköre, szerepe, a munkaerő kihasználásának hatékonysága és ideje [282, 283]. A fejlesztő-fejlesztő és fejlesztő-szoftvertermék közötti interakció analízise révén értékes információ gyűjthető amely felhasználható a termék minősítése és kockázatbecslés során. Yahav at al. javasolják, hogy ez a megfigyelés és elemzés folyamatos legyen, azaz a közösség állapotát folyamatosan monitorozzuk [284, 285], Smith pedig modell alapú megközelítést tanácsol [286].

A fejlesztők beazonosíthatósága komoly előnyt jelenthet komponens integráció során, hiszen közvetlenül azzal az emberrel lehet felvenni a kapcsolatot, aki az adott komponenst készítette[121]. Erre IP, üzleti titok és szervezési okokból kifolyólag vajmi kevés esély van az üzleti modell esetén. A vezető fejlesztőket a kommunikációban elfoglalt centralitásuk alapján lehet beazonosítani [287, 288].

A fig. 2.7a ábrán a Mozilla Firefox projekt fejlesztőinek komponensekhez rendelése látható, míg a fig. 2.7a ábrán ugyanezen projekt hibajegyekből generált fejlesztői centralitás vizualizációja figyelhető meg, amelyből kitűnik, hogy a bzbarsky@mit.edu és a asa@mozilla.org fejlesztők magas centralitás-értékekkel rendelkeznek (így az ábra közepe táján találhatóak), valódi nevük és kilétük egyszerű keresőhasználattal perceken belül kideríthető.



(a) Komponens-fejlesztő viszony (b) Mag periféria minta

2.7. ábra: Példa a szervezeti felépítés vizualizációjára. Forrás:[288]

Hasonlóképpen elemezhető a projekt fejlesztő dinamikája is, azaz projekt fejlesztő-megtartó és fejlesztő-vonzó képessége[289], melynek segítségével akár az egyes fejlesztők kilépésének valószínűsége is megjó-

solható [263].

2.5.1.4 Önszerveződés

A közösség egy célkitűzés, ideológia és kulcsemberek köré szerveződik[93]. A csoport sikere alapvetően a közösségtől, annak szerveződő-képességétől függ. Ha a nem sikerül kellő számú hatékony fejlesztőt vonzania, aligha lehet sikeres [290].

A tagok általában maguk választanak maguknak feladatot [51, 54, 60] azaz a feladatok kiosztása is önszerveződő. Nem lehet valakit utasítani vagy elbocsájtani, nincs fejlesztő felvétel és alkalmassági vizsga, ehelyett fejlesztő-bevonzás van [61]. A humán erőforrás menedzsment nem is hasonlít az üzleti világban megszokotthoz [54]. A kevéssé népszerű feladatok elvégzése kapcsán a belépő szervezetek szerepe felértékelődik, hiszen megfelelő (általában anyagi) motivációval kiegyensúlyozhatja az egyenlőtlenségeket. A munkamegosztás egy közösségen belül lehet jól definiált, vagy teljesen ad-hoc [57] jellegű is.

Gyakran megfigyelhető valamilyen mentori szisztéma, amelynek során egy régebbi tapasztalt fejlesztő segíti az újonnan beszállni kívánót. A mentorált csatlakozók mérhetően hatékonyabbak mint az önállóak [291], és a szociális akadályok leküzdésében is sokat segíthet egy jó mentor, annak híján pedig egy csatlakozni kívánóknak szóló dedikált honlap[292, 293].

Az önszerveződés nem jelenti azt, hogy nincs szükség menedzsmentre. A fejlesztői hálózat szervezését elhanyagoló projektek sokkal rosszabbul teljesítenek, gyakran meg is szűnnek [134]. A kisebb, önjelölt vezérrel rendelkező projektek legfeljebb akkor tudnak fejlődni, ha valamilyen módon fel tudják kelteni a figyelmet [86]. A nagyra növő, rosszul szervezett projektekben pedig a nehézkes kommunikáció megnöveli a feladatok lezárásához szükséges időt [294].

Érdekes megfigyelés, hogy a résztvevők döntő többsége csak kevés (gyakran egyetlen) projektben vesz részt, és csak nagyon kevesen ismerik az egész projektet [295, 296]. Ez arra enged következtetni, hogy a jó fejlesztőkért komoly a verseny, a nyílt közösségben szerepet vállalni kívánók száma[283] és a fejlesztésre szánható idő is véges. Következésképpen a nyílt közösség megtartása és vonzása érdekében érdemes bizonyos marketing tevékenységet végezni.

2.5.1.5 Döntéshozatal, irányítás, befolyás

A klasszikus modellben általában nincs világos vezetői lánc (FS-K-SZ), a döntések sok vitával járnak ami megnöveli a koordinációs erőfeszítéseket [203], a nagyobb projektek (talán épp emiatt) centralizáltabb döntéshozatalt alkalmaznak, ugyanakkor a döntések és indoklások szinte mindig átláthatóak maradnak (FS-K-Sz), máskülönben elidegenítik a fejlesztőket és leépítik a közösséget. Nincs alkalmazható kényszer, így bizonyos népszerűtlen feladatok elvégzetlenül maradnak, illetve a nagyobb, szponzorált projektekben az ilyen feladatokat fizetett fejlesztők végzik el[83]. Gyakori, hogy saját normarendszert fejlesztenek ami a fejlesztés irányát és a követendő ideológiát is kijelölti [255].

A közösség irányítása lehet decentralizált "Bazár" stílusú vagy centralizált, esetleg hierarchikus felépítésű. A előbbiek általában a független nem rutinszerű nagy bizonytalanságú feladatokban teljesítenek jól, míg a rutinszerű, erősen összefüggő, kis bizonytalanságú feladatokhoz a centralizált felépítés a megfelelőbb [297].

A vezetők – klasszikus esetben – kis részben hagyomány, nagyobb részben alkalmasságuk alapján kerülnek kiválasztásra többnyire demokratikus formában, így a klasszikus nyílt közösség meritokráciának vagy technokráciának tekinthető[93]. A bizalom kiépítésénél a legfontosabb tényezők a fejlesztői tudás, a reputáció valamit a formális és informális tevékenység a közösségen belül [102, 298]. A vezetők szerepe és kiléte meghatározó [267], egy-egy vezéralak véleménye nagy súllyal eshet latba a döntéseknél. A gitHubon belül például a GitHub csapat tagjai, a szervezetek, az OSSD fejlesztők és a keretrendszer/progamkönyvtár szerzők rendelkeztek a legmagasabb bizalmi szinttel [299]. A hatásgyakorlás inkább csak belülről lehetséges, ezért a szoftvercégek gyakran szponzorálnak fejlesztőket akik képviselik az érdekeiket [167, 300]. Aaltonen az e-mail címek elemzése alapján kimutatta, hogy az iparági szereplők befolyása meglehetősen nagy például a Linux kernel projektben [107] (jellemző szereplők: Redhat, HP, Oracle, Intel, IBM) vagy Debian operációs rendszerben [301] (SUN, Xerox, Digital Equipment corp, Silicon Graphics, HP stb.). A nyílt projektben való részvétel egyik kockázata az irányítás elvesztése [302], az új közösség kiépítése pedig igen költséges és időigényes.

A közösség fontos döntéseiben a mag csoport tagjainak sokkal nagyobb szerepe van, az egyéb módon fel nem oldható vitás kérdéseket is ők rendezik [303]. Emiatt fontos e szereplők beazonosítása, hiszen segítségükkel lehet leginkább hatni a csoportra, illetve ezeket a szereplőket érdemes a szervezetnek támogatni.

[256]. A döntéshozatal általában nem teljesen demokratikus, a mag résztől távolabb esőknek kevesebb beleszólásuk van a döntésekbe, illetve a mag élén állók gyakran fenntartják maguknak a jogot a végső döntésre. A vezérek sem tehetnek meg bármit, hiszen ha szembe mennek a közösséggel akkor annak elvesztését vagy a projekt másolását (fork) kockáztatnák. Legtöbb projekt esetében tehát a végső döntés közösségi nyomásra jön létre, nagyon sokat számít, hogy a változtatni kívánó fél mekkora lobbierőt tud felvonultatni, hány embert tud maga mellé állítani [80]. A változásokat egyáltalán nem olyan egyszerű keresztülvinni, mint az ember elsőre gondolná. Ennek oka, hogy a szoftver óriásira duzzadna ha bárki beletehetné kedvenc funkcióját (feature bloat), ami ellen a fejlesztői közösség általában keményen fellép [83].

Az OSSD menedzsment vonatkozásai merőben eltérnek az üzleti fejlesztéstől. Vannak megrögzötten független fejlesztők, akik igen nehezen viselik ha a projekt mögött egy nagyobb cég áll [253]. Emiatt fontos leheta szervezet (érzékelhető) jelenlétét a lehető legkisebbre csökkenteni. Az ipari szereplőknek viszont könnyen megérheti a közösséghez való csatlakozás, vagy saját közösség létrehozása, hiszen a közösség által, fejlesztőkhöz, informális tesztelőkhöz és rengeteg visszajelzéshez jutnak [145, 162] valamint növelhetik befolyásukat a projekt céljait illetően[300]. Bármilyen nagy is egy cég, annyi fejlesztőt egyszerűen nem bír megfizetni, amennyi egy komolyabb OSSD projektben összegyűlik[55, 304]. Következésképpen a szervezet felépítése gyakorta bővül fizetett fejlesztőkkel és iparági kapcsolatokkal.

A klasszikus OSSD nyitottsága természetszerűleg támogatja a belépést[305], a meritokrácia jelleg miatt a cégeknek gyakran nincs is más lehetőségük az irányításra. A belépés – mint láttuk – általában meg is éri, ugyanakkor adaptálódni kell az adott projekt kultúrájához[306] nagyon nehéz megváltoztatni azt. Nem előnyös például ha a saját fejlesztők külön csatornán kommunikálnak. A közösség átláthatóságot vár[44]. Nem javasolt továbbá a modern extreme programozási[25] technikák használata sem, amelyek zárttá teszik a csoportot [307].

Ha a cég jelenléte nagy, akvizíciója komoly veszélyt jelenthet a projekt jövőjére nézve és súlyos zavart kelthet a közösségen belül még akkor is, ha a terméket érintően semmilyen változtatás nem történik [308]. Hasonlóan súlyos lehet a helyzet ha a cég elhagyja a korábban vezetett projektet akkor is, ha korábban az nélküle működőképes volt [29].

[25] Extreme Programming. Modern szoftverfejlesztés módszertan az agile software development egyik ága. Jellegzetes technikái a páros programozás, extenzív kódvizsgálat, hierarchia nélküli menedzsment és a gyakori interakció az ügyféllel.

Amennyiben egy gazdasági szereplő hatást szeretne gyakorolni a közösségre, a következő lehetőségei vannak[309]:

- integráció: a szervezet beépül a közösségbe. Szerepe nem szükségszerűen vezető, a közösség által előállított FLOSS komponensekből származó haszon az együttműködés célja.
- közösség létrehozása: a szervezet maga hozza létre a közösséget. A kulcskérdés itt a közösség szervezése, de az integrációval ellentétben a szervezet saját üzleti stratégiájához illeszkedő erőforrásként kezeli a közösséget.
- hatalomátvétel: a szervezet egy létező OSSD közösség felett veszi át az irányítást céljait tekintve a létrehozással azonos.
- másolás (fork): a szervezet saját független változatot indít a FLOSS termékből. Ez általában akkor következik be ha az integrációt követően a közösség olyan irányt vesz fel amely a szervezet üzleti vagy stratégiai céljaival összeegyeztethetetlen. A forkolt közösség feletti irányítás átvétele lehetséges, de nem követelmény, hiszen a forkolt változatot követő fejlesztők oszthatják a szervezet nézeteit.
- kiadás: a szervezet nyílt forrásúként ad ki egy terméket, de nem foglalkozik vele, hogy épül-e köré közösség vagy sem. Ez a stratégia figyelhető meg többek között a kormányzati szektorban, ahol a kormányzati forrásokból készült terméket OSI kompatibilis licenccel adják ki.

A kód megnyitása vagy új közösség szervezése korántsem jelent garantált sikert. Könnyen előfordulhat, hogy a közösség nem nő, az újonnan érkezők egyetlen alkalom után továbbállnak, elvesztik érdeklődésüket vagy forkolják a projektet[310].

Schaarschmidt szerint az üzleti szereplők alapvetően kétféleképpen fejthetnek ki hatást a nyílt közösségekre[272]:

- *irányítás a vezetésen keresztül*: saját embereiket juttatják a közösség vezető pozícióiba vagy megfizetik az eleve ott található fejlesztőket;
- *irányítás erőforrás-bevitel segítségével*: vállalati környezetben szocializálódott fejlesztőket juttatnak a közösségbe, és klánkontroll segítségéve formálják annak szokásait. A későn belépő szervezet számára gyakran csak ez a lehetőség nyitott, mert csak nehezen tud vezető pozíciókat megszerezni.

A fizetett fejlesztők valószínűleg ténylegesen nagyobb hatást képesek kifejteni a közösségre, mindenesetre több emberrel kommunikálnak, velük is több ember keresi az interakciót[300] és a centralitási értékeik is

magasabbak [294]. Nem csak a cég számára előnyös a közösség, ez fordítva is igaz, a közösség helyezését is emeli a cégek belépése[43].

A nagy OSSD projektek irányítása sok tekintetben hasonlít a politikára, egzakt módon nehezen megfogható szociális vonatkozásai vannak [311], sok szereplő küzd különféle módszerekkel a befolyásért [80, 279] és néha ütköznek az érdekek. A projekten olyan vállalatok dolgozhatnak együtt, akik nyílt piaci környezeteben vetélytársak. Teixeira például megfigyelte, hogy a vállalatok gyakran beviszik saját ellentéteiket a közösségbe [312] amit a nyílt projekt mérhető sérüléséhez vezet. A vállalatok dedikált technikákat alkalmaznak, hogy belső, minősített információk ne szivároghassanak ki a nyílt együttműködésen keresztül [313]: a megosztott adatokat üzleti szempontok szerint szűrik és felelősöket "Gatekeeper''-eket jelölnek ki, akik a szervezet határán állva ellenőrzik az információ-áramlást.

2.5.2 Résztvevők (FS-K-R)

Mint kiderült a kulcsszereplők beazonosítása igen fontos, ez azonban nehézségekbe ütközhet. Nem csak a konkrét személy identitását lehet nehéz megállapítani, hanem az egyes pszeudo-anonim identitások (e-mail, nick, álnév, cím stb.) azonosságát is[314]. Néha a személyazonosság pár perces kereséssel kideríthető, más esetben – különösen ha a fejlesztő szándékosan rejtőzködik – a feladat közel lehetetlen. Jó példa erre Satoshi Nakamoto, a Bitcoin hálózat tervezője, akinek kilétét az amúgy jelentős erőfeszítések ellenére is mind a mai napig homály fedi. Még azt sem sikerült hitelt érdemlően megállapítani, hogy egyetlen személyről vagy egy csoportról van-e szó.

Nem triviális összevetni, hogy egy adott módosítást végző személy azonos-e egy másik módosítást végző személlyel, a program vezető fejlesztőjével vagy a hibajegyet kezelő személlyel. Ezen a problémán segíthet valamelyest a terjesztett csomaghoz csatolt vagy verziókezelő rendszerben, disztribúció csomagjaiban tárolt digitális aláírás. Sajnos sok esetben a fejlesztők több ilyen aláírást is használnak (elvesztés, frissítés) tehát a módszer korántsem száz százalékos és gyakran csak a pszeudo-identitáshoz rendelés lehetséges.

A résztvevők sokféle környezetből érkeznek, vannak közöttük kiváló szakemberek[83], hátterük igen eltérő mind életkorra, származásra és képzettségre való tekintettel [315]. Általában jó csapatjátékosok és alapvetően implementáció orientáltak [45].

Az üzleti fejlesztés profitvezérelt (külső motiváción alapul), a nyílt fejlesztést (elsősorban belső) motivációk hatásegyüttese jellemzi. Fejleszthet valaki azért mert tanulni akar, vagy hiányzik egy rég kívánt képesség a kedvenc szoftveréből, vagy egyszerűen az alkotás örömeért, vagy a társadalom iránti elkötelezettségből, altruizmusból[60]. A következő fő indítékcsoportok motiválhatják a fejlesztőket [38, 147, 316–318]:

1. Altruizmus (személyes hit a OSSD-ben, nyomot hagyni a világban, segíteni másokon)
2. Külső motiváció (karrier lehetőségek, előmenetel, szociális státusz, díjak)
3. Tudásbővítés (területi érdeklődés, új helyek, emberek, eszközök megismerése);
4. Közösségi,szociális indokok (csoporthoz tartozás, kommunikáció, tudásmegosztás)
5. Belső késztetés (munka öröme, önkifejezés, szórakozás, irányítás érzése)
6. Személyes igények kielégítése (hiányzó funkciók elkészítése);

A korábban belső indokok miatt fejlesztők motivációja nem változik meg akkor sem ha munkájukért fizetést kapnak [318].

A fizetett fejlesztők motivációja külső, fajtái az alábbiak lehetnek[319]:

- szabad szponzorálás, Nincs konkrét instrukció az alkalmazótól;
- alkalmazotti viszony, világos feladatokkal ;
- részlegesen kötött, a fejlesztő ideje egy részével rendelkezhet;
- adott cél eléréséért kapott díj, megbízási viszony.

A belső motivációból előnye, hogy erősebb mint a külső motiváció [254] hátránya viszont az irányíthatóság hiánya, így például a belső indíttatásból fejlesztők nem igazán kedvelik az adminisztrációt [320], az “unalmas” feladatokat [56], kedvelik viszont a nyíltságot, kényesek a kommunikáció minőségére, a termék nyíltságára [321] valamint a licencre [322].

A motiváció ismerete fontos, ugyanis ha a szervezet befolyást akar gyakorolni a közösségre saját delegált emberei által, a delegáltaknak tisztában kell lenniük a OSSD fejlesztők motivációival, ha tudni akarják hogyan hathatnak rájuk[323]. Az egyik érdekes lehetőség a gamifikáció, vagyis a repetitív, népszerűtlen feladatok játékos formába öntése [324].

2.5.3 Összefoglalás

A nyílt és zárt modell szervezeti felépítése és szociális struktúrája jelentősen eltér. A fejlesztőközösség szerkezete közösségi hálózat szerű, irányítása pedig magas technikai felkészültséget, ugyanakkor jó szociális érzékenységet illetve némi politikusi vénát igényel. A nyílt közösség nehezen befolyásolható, működése viszont teljesen átlátható, így az esetleges kockázat könnyebben becsülhető.

A FLOSS felhasználására vonatkozó biztonsági hatások csak közvetetten, a közösség működésén keresztül érzékelhetők. A közösség hatékony működése általában feltétele a hosszú távú működésnek, azaz a közösség segítése és irányítása, de legalább elemzése valamilyen módon szerepet kell kapjon az SCRM feladatokban.

Nagyobb volumenű felhasználás esetén elengedhetetlen a projekt befolyásolása, amelyet csak a közösségben való részvétel által érhető el. A részvétel történhet közvetlenül szervezeti keretek közt illetve állami szinten vagy alapítványok formájában.

A fejlesztők pontos kiléte nem mindig állapítható meg, de a pszeudo identitások katalogizálása is megoldható, még ha nem is szokványos gyakorlat.

2.15. táblázat: A K kategóriában azonosított problémák

kód	szint	leírás	sajátosság
SK01	1	A fejlesztők és a hibabejelentők közötti kommunikáció nem megfelelő.	FS-K-Sz
SK02	1	Az alkotók nem mindig beazonosíthatók. Nehéz megállapítani, hogy több részmunkát ugyanaz a személy végezte-e el vagy sem.	FS-K-Sz
SK03	1	Az erős cégirányítás miatt a fejlesztők elhagyják a projektet, veszélyeztetve a támogatást	FS-K-Sz
SK04	4	Rossz vagy az elvárásoknak nem megfelelő kód küldése a bizalom, ezáltal az OSSD feletti irányítás elvesztését okozza.	FS-K-Sz
SK05	1	Az OSSD vezető szereplője távozik (cég akvizíciója, fejlesztő kilépése), a támogatás folytonossága veszélybe kerül.	FS-K-Sz
SK06	4	A delegált fejlesztők nyílt kommunikációja érzékeny adatokat tesz publikussá.	FS-K-Sz

2.16. táblázat: A K kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JK01	4	A közösségbe fejlesztőket kell delegálni akiknek lehetőség szerint mentort kell keresni.	SJ01
JK02	2	A belső szerkezet elemezhető, meghatározhatók a kulcsszereplők. Az adatok alapján kockázatelemzés végezhető.	SF07, ST02
JK03	2	A digitális aláírásokat használó fejlesztők nyílt kulcsait (rendszerint GPG) be kell szerezni, biztonságos adatbázisban kell tárolni, hozzáférhetővé tenni és szükség esetén ellenőrizni.	
JK04	2	Folyamatosan monitorozni kell a közösséget.	SK05
JK05	2	Megfelelő lobbierőt kell kiépíteni az érdekérvényesítő képesség érdekében.	SF07, SH07, ST02
JK06	2	Kapcsolatépítéssel és személyes találkozásokkal erősíthető a közösséggel való együttműködés (pl.találkozók szervezése).	SF07, SH07, ST02
JK07	2	A vezetőkben tudatosítani kell, hogy az OSSD fejlesztőit más módon kell motiválni.	SK03, SK04, ST02, ST03
JK08	4	Felelőst kell kijelölni (Gatekeeper) aki a közösséggel való kapcsolattartást és az oda áramló információt ellenőrzi.	SF07, SF14, SK06, SM05
JK09	4	A közösségbe delegált fejlesztők elkülönített kommunikációja előnytelen, a közösségi csatornán folyó kommunikáció információtartalmára oda kell figyelni.	SK06

2.6 Metaadatok (FS-M)

A nyílt és zárt fejlesztések egyik nyilvánvaló különbsége a rendelkezésre álló nagy mennyiségű fejlesztési információ. Elérhető például a belső dokumentáció, a forráskód, a viták archívuma és minden hibajegy illetve azok javítási folyamata. Ezeknek a metaadatoknak a léte, létrehozása, kezelése és vizsgálata hatással lehet az alkalmazó vagy integrátor rendszerének biztonsági szintjére is.

A nyílt fejlesztések során a közösség tagjai között zajló információcsere négy jellemző információs térben valósul meg:

- egyeztetési tér, amely magába foglalja a levelező listákat, webportálokat, fórumokat, leveleket és blogokat;

- dokumentációs tér, amely magába foglalja a dokumentációt, a webes forrásokat és verzió kezelő rendszerek metaadatait;
- karbantartói tér, amely a hibajegy kezelő rendszerekben tárolt információt jelenti;
- és implementációs tér, azaz maga a forráskód az itt található megjegyzésekkel együtt.

Viorres, Mistrik és Tuunanen [325–327] a karbantartói teret a dokumentációs tér alá sorolják, véleményem szerint azonban a hibajegy követés során alapvetően más jellegű kommunikáció (felhasználó - fejlesztő) zajlik, más célokkal és hangsúllyal, egyfajta átmenetet képezve a dokumentációs és implementációs tér között.

2.6.1 Karbantartói tér, hibajegyek, hibakommunikáció (FS-M-H)

A nyílt forrású projektek jellemzően használnak valamilyen hibakövető (bugtracker) rendszert. A nagy projektek esetében ez arány 96%[127] míg a közepes és kisebb projektek esetében valamivel kevesebb. Akárcsak a projekt többi része ez is nyílt, az itt tárolt adatok és kommunikáció könnyen hozzáférhető és elemezhető.

Számos szabad hibakövető rendszer áll rendelkezésre (Mantis, Redmine, BugZilla stb.) illetve a népszerű közösségi fejlesztőplatformok (GitHub, GitBucket) beépítetten is biztosítanak ilyen lehetőséget. Persze a hibakövető rendszer használata önmagában nem egyedi sajátosság, hiszen a zárt fejlesztések, sőt szolgáltatók is alkalmaznak valamilyen vagy éppen valamelyik fent felsorolt hibakövető rendszert. A FLOSS-ra jellemző egyedi jelenség viszont, hogy ezeket a rendszereket bárki – komolyabb authentikáció nélkül – használhatja[328]. A használat módja, a felvitt hibajegyek és azok hasznosulása, kivált pedig a teljes információmennyiség folyamatos átláthatóság az, ami megkülönbözteti a nyílt közösséget [55, 135].

A szabadon elérhető hibajegy információ automatizált módszerekkel elemezhető[290, 329, 330], adatbányászati technikákkal, kulcsszó alapú kereséssel értékes információkat kaphatunk a projekt állapotáról, hibajavító képességéről, szociális szerkezetéről [331]. A hibajegy-kezelőből kigyűjtött információkat összevethetjük sérülékenység adatbázisok adataival [332], illetve alkalmazhatunk egyedi modell alapú hiba-előrejelző rendszert [333, 334] vagy hagyományos szoftver megbízhatósági modelleket [335]. A publikus hibajegyek negatív hatása, hogy a sérülékenység mindenki számára azonnal elérhetővé válik, így a sérülékenységet elvben kihasználni képes potenciális támadók köre is szélesebb.

Az hibajegyekből néhány általános, a nyílt fejlesztésre jellemző biztonsággal kapcsolatos paramétert is megtudhatunk. Az alábbiakban ilyen adatokat mutatok be röviden kifejtve annak esetleges biztonsági hatásait.

A hibajegyeket általában (67%) konzisztensen kezelik de a megoldatlan hibák száma 20% körüli[127]. A minket érintő hibák tehát nem feltétlenül lesznek kijavítva, így fontos a részvétel vagy a megfelelő irányítási és/vagy motivációs lehetőség kiépítése. A sérülékenység száma az idő előrehaladtával általában növekszik (Apache, Mozilla, Linux) [336–338], ami első hallásra nem vet túl jó fényt a nyílt fejlesztésekre, bár lehetséges, hogy a hibaszám növekedése nem a minőség romlását jelzi, egyszerűen csak arról van szó, hogy a felhasználók és a fejlesztők egyaránt egyre biztonság-tudatosabbá válnak és jobban odafigyelnek [338]. A biztonsági bejelentéseket általában más elbírálásban részesítik és gyorsabban javítják őket [339].

A felhasználói interfésszel kapcsolatos visszajelzések általában hamar megérkeznek[340], ami várható is hiszen ezzel a résszel találkoznak leghamarabb a felhasználók. Bizonyos hibák felderítése és különösen reprodukálása ugyanakkor igen nehéz lehet (pl. concurrency, race condition) [336, 338] ha pedig a reprodukálás valószínűsége alacsony, a fejlesztők elutasíthatják őket. Nagy figyelmet kell fordítani tehát a pontos és minél jobb hibakeresési lehetőséget nyújtó hibajegyek létrehozására, ami különösen nehéz, ha a kiadható gépi információk halmaza nem tisztázott.

Mivel a nyílt közösségekben nem lehet “kérni a panaszkönyvet”, kötelezettség szegési eljárást indítani, de általában reklamálni sem nagyon érdemes, potenciálisan fontos lehet, hogy megtaláljuk azokat a módszereket és személyeket akik meg tudják oldani a problémáinkat. Minél magasabb kapcsolati szinten (lásd. FS-K-SZ) áll egy fejlesztő annál hamarabb javítja a hibákat, viszont minél többet kommunikál, annál lassabban [341].

A hibabejelentések jelentős része egyszerű segítségkérés, valódi értéke nincs így zajnak tekinthető [342]. Az értékes hibajegyeket egy szűk felhasználó kör veszi fel, tehát nem csak viszonylag jól definiált fejlesztői és hibajavító csoport, de jól elkülönülő hibakereső halmaz is létezik. Felmerül a kérdés, hogy ezek után a nyílt forrásnál gyakran felhozott “many eyes” elv – azaz, hogy kellően sok szem minden hibát észrevesz[60] – mennyiben különbözik egy nagyobb vállalat széles körű béta tesztjétől. Az adatok tanúsága szerint (legalábbis az elemzett Mozilla szoftver esetében) úgy néz ki nem sokban [330]. (Amint azt a fejlesztésről szóló szakaszban kifejtem (FS-F-P), a meglepően jó teljesítmény nem is a tesztelők számának hanem az

egyedi módszertannak köszönhető).

A sok éles szem elvvel szemben más kétség is felmerül. A nagyobb nyílt közösségnek általában van saját biztonsági csapata. A biztonsági csapat méretétől elvben függenie kéne a hibajavítási teljesítménynek, ez azonban statisztikailag nem igazolható (Mozilla, Tomcat, Apache)[337]. Valószínű, hogy az adott kódrészletet kevesen értik meg vagy kevesen szánják rá az időt, így a hibás részt végül többnyire az eredeti szerző javítja ki.

A kritikus és nem kritikus hibák javítási idejében jelentős eltérés mutatkozhat. A nem kritikus hibák javítása igen nagy mértékben akár évekig is csúszhat, míg a kritikus hibákat a legtöbb közösség 1-3 napon belül javítja [135]. A hibajavításnál a hiba ellenőrzése tart a legtovább (azaz míg a hibajegy "resolved" állapotból "resolved bug verifed" állapotba kerül) [342] ami arra enged következtetni, hogy a hangsúly a hiba kijavításán és nem a minőségellenőrzésen van, ezt inkább a közösségre bízzák. Ennek ellenére a hibajavítások általában jók, nincs szükség a hibajavítás javítására [337], ugyanakkor Francalanci úgy találta, hogy a hibajavítások minősége rendkívül változó[132]. A hibajegyek státuszának állítására általában igen sok embernek van lehetősége ami minőségi problémákhoz vezethet [328].

Végezetül érdemes megjegyezni, hogy bár sokan elsőként a honlapokon feltüntetett letöltésszámot vagy megtekintések számát nézik ha megbízható projektet keresnek, ez egyáltalán nem jó mérőszám. Nem mutatható ki összefüggés a hibaszámmal és nem alkalmas a projekt minőségének becslésére [180].

2.6.2 Dokumentációs tér, architektúra és API dokumentáció (FS-M-A)

Tekintettel a nyílt modell megoszló közösségére és fejlesztési módszerére, logikusan azt várhatnánk, hogy ha egynél több, egymást kevéssé ismerő fejlesztő dolgozik együtt, létfontosságú a jó minőségű fejlesztői dokumentáció léte. Meglepő módon ez nem mindig van így és legfőképpen nem minden téren teljesül[91].

Magát a kódot többnyire elegendő megjegyzéssel látják el, a megjegyzések sűrűsége független a projekt vagy csapatmérettől[343], de a jó minőségű dokumentáció létrehozása már kevéssé érdekli a fejlesztőket [98, 185] ami valószínűleg a világos szervezeti felelősségek hiányára vezethető vissza, illetve, hogy a fejlesztők az olvasható kódot (vagy abból generált dokumentációt) megfelelő dokumentációnak tekintik[55]. A tudás tárolására jellemző az automatikusan generált dokumentáció és a wiki forma [37], de előfordul,

hogy a résztvevők egy levelező listát használnak dokumentációnak, ami a projektbe később beszállók számára felettébb kedvezőtlen, hiszen egy adott funkció leírásának megtalálásához nagy számú téma-szálat kell végigolvasni[344]. Természetesen a nagy projekteknél ahol több száz vagy ezer fejlesztő dolgozik együtt és a felelősségek rendszere is világosabb, ez a probléma ritkán jelentkezik, ami érthető, hiszen világos fejlesztői dokumentáció nélkül egy ekkora projekt egyszerűen életképtelen lenne.

Az üzleti kódhoz képest (egy eset, megnyitott kódú RDBMS elemzés alapján) a nyílt fejlesztés során készült új részek átlagos komplexitása és függvényenkénti hatékony kódsor mutatója kisebb, szignifikánsan kevesebb megjegyzés sort használtak[345], ami arra enged következtetni, hogy a nyílt fejlesztők sokszor inkább a kódot tanulmányozzák a megjegyzések helyett. Minthogy egy-egy kódrészletnek nincs dedikált tulajdonosa és a fejlesztés kevésbé szervezett előfordul, hogy azonos komponensek több különféle leírással rendelkeznek [98] ami rontja a dokumentáció minőségét.

Másik fontos kérdés az architektúra dokumentáció megléte, amely segít átfogó képet adni a teljes projekt struktúrájáról és belső összefüggéseiről. Mint kiderült a FLOSS projektek architektúra dokumentációja sok esetben – akár nagyobb projektek esetében is – kifejezetten gyenge, elévült, esetenként teljesen hiányzik és a szükséges információt más dokumentációkból illetve a kód átnézésével kell kikövetkeztetni[37].

Biztonsági szempontból ez azért aggályos, mert a sérülékenységeknek csak egy része származik közvetlen kódhibákból, a hibás tervezési döntésekből adódó (esetenként súlyos) biztonsági problémák csak architektúra dokumentáció alapján lennének hatékonyan beazonosíthatóak[346] míg kellő erőfeszítéssel egy eltökélt támadó a nyilvános információk alapján feltárhatja ezeket. Természetesen a jelenség nem általános vannak nyílt projektek ahol a szerkezeti felépítés világosan definiált, hiszen megléte fontos lehet a projekt túlélése szempontjából [241], a könnyen érthető, jó szerkezeti diagramokkal rendelkező projektek pedig több fejlesztőt vonzanak [87].

Összefoglalva, a projekt integrációja vagy a szoftver minősítés során érdemes figyelembe venni a meglévő dokumentáció minőségét, míg a projektben való részvétel esetén a megfelelő dokumentáció kialakítására és a dokumentálással kapcsolatos feladatkörök tisztázásra kell különös gondot fordítani.

2.6.3 Egyeztetési tér, a fejlesztői kommunikáció vizsgálata (FS-M-K)

A nyílt fejlesztés jellemzője, hogy nyomon lehet követni a teljes fejlesztői tervezési folyamatot, olyan – egyébként általában privát – információk is nyilvánosan elérhetőek, mint a fejlesztők egymás közötti kommunikációja. Ez az adathalmaz szintén elemezhető, vizsgálható a fejlesztők reakció ideje, gyakorisága vagy akár a csoport stabilitása is [105, 347–349].

Az egyeztetési tér tárolási platformja igen változatos. Lehet kifejezetten dedikált célszoftver, wiki platform, nagyon gyakori forma a levelező lista, elemezhető információt hordozhatnak a blogok és a mikroblogok (pl. Twitter) [350].

A kommunikáció statisztikai elemzésével sikeresen elemezhető a projekt fejlődése [45], megjósolható a fennmaradás esélye [351, 352], ugyanakkor a várakozással ellentétben a projekt életciklusának végét nem lehet biztonsággal meghatározni [353]. Következtetéseket lehet levonni viszont a fejlesztők elkötelezettségére és a támogatói tevékenységre vonatkozóan [354].

Megfigyelhető, hogy a tudásmegosztás egy része a levelező listákról áttevődik a dedikált Kérdés-Válasz kezelő oldalakra (például StackExchange) [324]. Minthogy ez az anyag is publikus, manuális átnézése vagy automatikus elemzése szintén segíthet képet alkotni a projekt állapotáról.

2.6.4 Implementációs tér, a forráskód elemzése (FS-M-I)

Az elérhető forrás természetszerűleg lehetővé teszi, hogy a kódminőséget statikus white-box módszerekkel elemezzük [355–359], ami eleve eltérés a zárt modellhez képest, ahol ez általában nem lehetséges. A nyílt forrás statikus kódelemzése igazoltan csökkenti a biztonsági hibák számát, szignifikáns korreláció mutatható ki a statikus analízis végzése és a CVE sérülékenységek előfordulása között [360].

Amennyiben az kód valamilyen szabvány implementációja az ellenőrizhetőség követelmény is lehet. Tiemann szerint az “implementáció nélküli szabványok megbízhatatlanok, azokat az implementációkat, amelyeket nem lehet átvizsgálni, ellenőrizni sem lehet” [361]. Ugyanígy követelmény lehet a nyílt implementáció a trusted computing területén, azaz mindenhol ahol felmerülhet a felhasználók utáni “kémkedés” kérdése. Különösen fontos lehet ez a LRE területén, ahol magas biztonsági követelményeknek kell meg-

felelni. Minden ilyen esetben erősen javasolt a TCG[26] kompatibilis nyílt implementáció használata[139, 362].

A vizsgálat történhet manuálisan egyénileg, ez azonban jelentős szaktudást igényel, ezért a normál felhasználók számára a kód elérhetőség nem jelent semmiféle előnyt, gyakorlatilag ugyanúgy használják, mint a normál COTS termékeket [70]. Léteznek viszont megoldások kifejezetten a kód folyamatos vizsgálatára [363]. Ilyen például az amerikai kormányzati támogatással 2006-ban indult Coverity Scan amely "az egyik legnagyobb közösségi-kormányzati projekt a világon, melynek fókuszában a nyílt forrású szoftver minősége és biztonsága áll"[27]. A regisztrált projekteket folyamatosan nyomon követik és a kódot statikus elemzés módszereivel rendszeresen ellenőrzik, a felderített hibákról pedig jelentéseket küldenek. (A OSI elveinek megfelelő projektek regisztrációja ingyenes.)

A forrás-elemzés segíthet megítélni a projekt stabilitását és fejlődési lehetőségeit [118]illetve a becsült munkaóra értékét[364], ám a rendkívül nagy számú változat léte nehézséget jelenthet. Ebben nyújthatnak segítséget a nyílt projektkövető rendszerek, amelyek nagy mennyiségű fejlesztői információt tárolnak. Ilyen például az Open Hub[28] (korábbi nevén Ohloh) vagy a kifejezetten kódanalízis célzatú FLOSSmole. Az itt összegyűjtött adatok által tömegesen is vizsgálhatóvá válnak a FLOSS projektek [365, 366].

A forrás elérhetőségének egy további előnye, hogy sokkal nehezebb elrejteni a biztonsági hibákat. Egy képzett fejlesztő a forrást átfutva részletes elemzés nélkül is képet kaphat a termék általános minőségéről[367]. (lásd még: FS-J-Á).

Elvben a közösség által módosítható forrás a kritikus sérülékenységek elhárítási idejét is lecsökkentheti, tehát a támadásra alkalmas időablak rövidebb, hiszen a támadónak nem egy szűk csapattal, hanem egy sokkal nagyobb fejlesztői csoporttal kell versenyeznie [139]. Ez azonban nehezen mérhető, ráadásul a fejlesztők száma önmagában nem sokat mond a résztvevők motivációjának és szaktudásának (FS-K-R) valamint a projekt frissítési sűrűségének ismerete nélkül (FS-T-K).

A nyílt forráskód logikus támadási felületnek tűnik, ám Erturk szerint – legalábbis a mobilalkalmazások piacán – a sérülékenységeket a támadók általában nem a forráskódban fedezik fel, hanem kívülről, pró-

[26] Trusted Computing Group, az AMD, IBM, Hewlett-Packard, Intel és Microsoft összefogásával indult csoport- amelyek célja a személyi számítógépeken alkalmazandó trusted computing elvek kutatása és definiálása.
[27] https://scan.coverity.com/
[28] https://www.openhub.net/

bálkozással, akárcsak a bármely más esetben [2]. Bár ez megnyugtatóan hangzik, egyáltalán nem zárható ki, hogy a kellő erőforrásokkal rendelkező támadó (pl. nemzetállamok kiberhadserege) a forrás elemzésével igyekezzen zero-day sérülékenységeket találni, ahogy az sem elképzelhetetlen, hogy a támadó fél szándékosan beépül a fejlesztői közösségbe, hogy ott álcázott sérülékenységeket helyezzen el (FS-K-SZ).

Az átláthatóság biztonsági kockázatot is hordozhat. A felfedezett sérülékenységeket morális okokból általában nem szokás felfedni amíg ki nem javítják azokat. Csakhogy a nyílt fejlesztés kódforrásának minden változása azonnal látható, így ha a hibát javítani igyekvő fejlesztők publikálják próbálkozásaikat, a sérülékenység automatziált módszerekkel lokalizálható. A hibajavítások átfutási ideje meglehetősen nagy – akár 100 nap is lehet – ami bőséges időablakot nyújt a támadás kivitelezésére. Yang at al. négy projekt (Linux kernel, OpenSSL, phpMyadmin, Mantisbt) automatizált elemzésével a 227 publikált sérülékenységből 140-et sikkerrel azonosítottak ilyen módszerrel, ami riasztó eredmény[368].

A legmagasabb biztonsági követelmények teljesítését igénylő feladatok esetében a szervezet formális bizonyítást lehetővé tevő nyelveken írt nyílt forrású kódot is alkalmazhat. Ilyen nyelv például a coq vagy az Agda [369]. Valójában épp a formális ellenőrzés lehetősége az ami a nyílt modellt legnagyobb előnye lehetne, sajnos azonban a kihasználásához szükséges tanulási görbe igen meredek, azaz mind fejlesztői mind felhasználói oldalon jól képzett szakemberekre van szükség . Jó példa a formálisan ellenőrizhető, biztonsági szempontból igen hasznos nyílt projektre a COMPCERT, ez magas biztonságú (C99) fordítóprogram, amellyel tanúsítható, mégis hatékony kód generálható[29][370]. További lehetőség olyan programozási nyelv használata, amely garantálni tudja a típusok és osztályhatárok megbízhatóságát. Nem tévesztendő össze a Type Safety tulajdonsággal, ami sok népszerű nyelvnek sajátja (többek közt a Java nyelv is ilyen). A Type Safety önmagában nem nyújt garanciát a rosszindulatú fejlesztői használat ellen, csak abban az esetben, ha a nyelv emellett garantálni tudja, hogy a futtatott – potenciálisan kártékony – kód semmilyen módon nem tudja áthágni a típusok és modulok általl felállított szabályrendszert. Ilyen nyelv például a SafeHaskell [371].

A nyílt közösségekre jellemző a verziókezelő rendszerek (git, svn, mercurial stb.) általános és magas fokú használata[53]. A projektek többsége (>80%) nemcsak tárolja, de rendszeresen frissíti is az itt összegyűjtött információt[127]. Az utóbbi évtizedben a forráskód tárolása részben áttevődött az kifejezetten ilyen céllal készült portálokra (GitHub, BitBucket, GitLab stb.) amelyek az eredeti funkciók megtartása mellett további

[29] http://compcert.inria.fr/

kiegészítő szervezési és kommunikációs lehetőségeket nyújtanak (pl. fórum, bug-tracker, pull-requests).

Az elérhető forrás segíthet tisztázni a felelősség kérdését is. Üzleti COTS alkalmazások használata esetén egy komplex rendszerben nehéz bizonyítani, hogy a hibajelenség valóban az adott szoftver vagy komponens hibás működéséből származik. FLOSS használata esetén ez mindig igazolható [70, 244].

Mint látható a forrás nyíltsága biztonság szempontjából a korábban említett "több szem többet lát" elven messze túlmutat, kritikus alkalmazás esetén pedig jelentős előnyöket biztosít. Különös módon az IT döntéshozók közül viszonylag kevesen tulajdonítanak neki megfelelő jelentőséget[239].

2.6.5 Összefoglalás

A FLOSS rendszerek és az üzleti termékek legfőbb eltérése a metaadatok kapcsán a nyíltság. Szabadon elérhető a forráskód, a rendszer API dokumentációja, a rendszerterv, a hibabajelentések, az azokra kapott válaszok, sőt sok esetben a hibák javítására tett kísérletek is.

Ez az átláthatóság néhány egyedülálló lehetőséget biztosít és egyben biztonsági kockázatokat is okoz. A metaadatok alapján kvalitatív vagy akár kvantitatív mérőszámokkal jellemezhető a projekt és a közösség minősége, készültségi foka, reakcióideje és általános állapota. Beazonosíthatóak a kulcsemberek, az egyes kódrészeket fejlesztők személye. Megfelelő keretrendszer esetén a teljes forráskód működésének helyessége formálisa igazolható. Ezzel szemben a hibajegyeken vagy a forráskódon keresztül érzékeny információ válhat publikussá, a forrás és a hibajegyek elemzésével még javítás előtt felderíthetőek a sérülékenységek, a forrás manipulálásával az összes felhasználó támadható.

A hibajegyek analízise alapján elmondható, hogy a hibajavítás folyamatához másként kell hozzáállni, mint az üzleti alkalmazások esetében, a hibajegyek nem megfelelő felvitele esetén azok javítása késhet vagy el is maradhat.

2.17. táblázat: A M kategóriában azonosított problémák

kód	szint	leírás	sajátosság
SM01	2	A hibajegy válasz nélkül maradhat.	FS-M-H
SM02	2	A hiba javításának ellenőrzése késhet.	FS-M-H

kód	szint	leírás	sajátosság
SM03	2	A hiba nem reprodukálható ezért figyelmen kívül hagyják	FS-M-H
SM04	1	A forrástároló, a publikus hibajegyek és a hozzá tartozó foltok folyamatos analízisével a támadó sérülékenységeket fedezhet fel és használhat ki még azok javítása előtt.	FS-M-H
SM05	2	A hibajegy hibakeresési csatolmányaként érzékeny információ kerül a publikus hibakereső rendszerre.	FS-M-H
SM06	1	A nyilvános fejlesztői dokumentáció tervezési hiányosságokat tárhat fel.	FS-M-A
SM07	1	A fejlesztői kommunikáció elemzésével a támadó még javítás előtt tudomást szerezhet a sérülékenységről.	FS-M-A
SM08	1	A kód analízise megkönnyíti a támadó dolgát (az elvi lehetőség adott)	FS-M-I
SM09	1	A támadó fél a forráskódban rejtett sérülékenységet helyez el.	FS-M-I
SM10	4	Nem világos hogyan kerül összhangba a forráskódban tárolt személyi adatok kérdése az adatvédelmi szabályozással (pl. GDPR).	-
SM11	1	Az architektúra dokumentáció sokszor hiányos, így a tervezési hibákból adódó sérülékenységek nem nyilvánvalóak.	FS-M-A

2.18. táblázat: A M kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JM01	1	Automatizált módszerekkel elemezhetőek a hibajegyek adatai	SF05, SF13
JM02	2	Be kell azonosítani azokat a fejlesztőket, akik hatékonyan javítják a hibákat.	SF13, SM01, SM02, SM03, ST02
JM03	2	Be kell azonosítani és követni azokat a hibakeresőket, akik valóban értékes hibajegyeket vesznek fel.	SF13
JM04	2	A hibakezelésben és fejlesztői kommunikációban aktív módon részt kell venni, az információáramlás gyorsítása végett.	SF13, SM01, SM02, SM03, SM07, ST02
JM05	1	Kódminőség vizsgálati módszerekkel elemezni lehet az elérhető forráskódot, ami szolgáltatásként is igénybe vehető.	SF05, SM04, ST05
JM06	3	A kód alapján eldönthető, hogy a hiba valóban az adott rendszerből származik-e vagy sem.	

kód	szint	leírás	probléma
JM07	3	A kód alapján bizonyítható, hogy nem történik illegális adatgyűjtés (TCG kompatibilitás)	
JM08	4	Résztvétel esetén megfelelő szintre kell hozni a dokumentációt és tisztázni kell a feladatköröket.	SJ01
JM09	1	Elemezni kell a metaadatokat, vagy ilyen elemzést lehetővé tevő minősítő keretrendszert kell igénybe venni	SF05, SJ01, ST05
JM10	3	A magas biztonsági követelményű feladatokra a szervezet csak formálisan ellenőrizhető nyílt forráskódot használ.	SM08, SM09
JM11	1	Forráskód és a hibajegyek elemzésével megítélhető a projekt stabilitása és fejlődési lehetőségei.	SF05, SF07, SF13, SH02, SH03, SH04
JM12	1	Forráskód átnézésével a kódminőség közvetlenül becsülhető, az alacsony minőség nem rejthető el.	SF05, SM08
JM13	2	A hibajegyeket már a kezdetektől jó minőségben, reprodukálást lehetővé tevő módon kell felvinni	SM01, SM02, SM03

2.7 Szabályozás, megfelelőség (FS-Sz)

Az alfejezetben a FLOSS szabályozással kapcsolatos kérdéseit tárgyalom. Szabályozás alatt itt az angol “compliance” kifejezésnek megfelelő fogalmat értem, tehát a külső és belső szabályoknak, irányelveknek, házirendeknek és jogszabályoknak való komplex megfelelést.

A megfelelőségi keretrendszerek (COBIT) illetve szabványok (NIST, ISO27000) korlátozott használhatósága a minősítő rendszerekkel rokon problémakör, amellyel a Felhasználás fejezet alatt foglalkozom (FS-F-M).

2.7.1 Belső szabályozás (FS-SZ-B)

Talán a mikrovállalatokat kivéve minden szervezetnek, legyen az kicsi vagy nagy, szüksége van valmilyen belső IT szabályozásra részben a jogi elvárások teljesítése végett, részben mert csak így biztosítható az működtetéssel járó kockázat elfogadható szintre csökkentése és az elvárt biztonsági szint fenntartása.

Különösen fontos ez a Létfontosságú Információs Rendszerelemek területén, ahol jogszabályban rögzített szigorú technológiai és szervezési elvárásoknak kell megfelelni.

Ezek a belső szabályzatok gyakran figyelmen kívül hagyják a FLOSS egyedi jellegzetességeit, egyáltalán nem, vagy csak ritkán szabályozzák részletesen a szabad szoftverekkel és nyílt forrású komponensekkel történő kapcsolattartás módszereit, ami biztonsági és üzleti kockázatot jelenthet.

Minthogy a FLOSS tekintetében jogszabályok nem írnak elő kötelező belső szabályozást, a definiálás során döntő szempont a menedzsment tudatosságának kérdése [140]. Sajnálatos módon kevés a FLOSS orientált szabvány, a komponensekkel kapcsolatos nagy mennyiségű, részleges, diverz és szubjektív információ nem igazán segíti elő a tudatos döntést [211].

Kemp szerint [372] a nyílt forrású szabályozás célja – más IP jellegű szabályozáshoz hasonlóan – a felesleges viták elkerülése, vásárlói elégedettség biztosítása és a források megfelelő menedzselése. A FLOSS szabályozás öt területet kell lefedjen:

1. Beszerzés: milyen FLOSS rendszereket használ a szervezet;
2. Megbízható források: ismerni kell honnan származik a FLOSS;
3. Követés: ismerni kell, mit csinál, hol használjuk és hol hasznosítjuk újra;
4. Szerepek és felelősség: tudni kell kinek mi a szerepe és miért felelős a FLOSS kapcsán; és
5. Licensz megfelelőség: ismerni kell, hogy a szervezet megfelel-e az OSS licencekben megjelenő kötelezettségeknek.

Ideális esetben a szabályzat kitér a:

- személyi kérdésekre, egyértelműen megfogalmazva az egyes résztvevők szerepköreit és céljait ;
- felállít egy követendő globális stratégiát, ide értve az üzleti, kockázatkezelési és IP stratégiákat ;
- foglalkozik a házirend kérdésével, amelynek tömörnek, egyértelműnek esemény orientáltnak kell lennie, kritériumokat és döntési pontokat kell megfogalmaznia az FLOSS felhasználása terén, és egyértelművé kell tennia gyűjtendő és követendő információkat;
- definiálja a folyamatokat, ahol meg kell fogalmazni a más területek szabályozásával kapcsolatos függőségeket, kitér a projekt-tervezéssel, kivitelezéssel és utókövetéssel kapcsolatos kérdésekre, illetve érdemes megfontolni a FLOSS közösség bevonását és megnyerését célzó alfolyamatok létrehozását

is;

- foglalkozik a sérülékenységek kezelésével.

Sajnálatos módon ezeket az elveket kevés szervezet követi a gyakorlatban. A legtöbb esetben ad-hoc módon hoznak döntéseket, pragmatikus, használat orientált módon amely nem vesz figyelembe sem költséghatékonyságot sem más magasabb rendű stratégiát [173].

2.7.2 Összetett licencelés (FS-SZ-L)

A FLOSS licenckörnyezete meglehetősen összetett. Számtalan szabad licenctípus létezik, amely több vagy kevesebb korlátozást tartalmaz illetve eltérő jogokkal ruházza fel a felhasználót. A szervezetek számára legvonzóbb ilyen jog a jogátruházás lehetősége, amely lehetővé teszi, hogy az üzleti licencek hagyományos EULA[30] szakaszával ellentétben a terméket ne csak használhassuk hanem lemásolhassuk és akár el is adhassuk [150, 373]. Az OSI[31] jelenleg (2018 augusztusa) 83 OSS licencet listáz, amely segítséget nyújthat a különféle licencek közötti eligazodásban [123]. A licenceket két alapvető csoportra, a megengedő (Permissive) és kötött (Restrictive) licenctípusok körére szokás bontani. Mindkét licenccsoport tagjai tartalmazzák a nyílt szoftverekre jellemző alapvető jogokat – a forrás elérhetőségét, szabad módosíthatóságát és a jogátruházás képességét – a korlátozások tekintetében viszont más feltételeket fogalmaznak meg. Leegyszerűsítve a megengedő licencek nem tartalmaznak korlátozásokat a komponenst felhasználó termék licencelésére vonatkozóan (pl. BSD), míg a kötött licencek elvárják, hogy a származtatott termék maga is kompatibilis licenccel rendelkezzen (pl. GPL) ami alól mentességet jelenthet, ha a származtatott termék nem az eredeti termék feladatkörét látja el, pusztán annak funkcionalitását használja (pl. LGPL).

A kötött licenceket "vírus szerűen terjedő" tulajdonsága miatt sokan bírálják[302, 374], támogatói szerint viszont éppen ezen tulajdonsága nyújtja a társadalom számára a legnagyobb előnyöket. A 2001-es évig a kötött licencekkel bíró termékek előretörése figyelhető meg, onnantól viszont – feltehetően a piaci viszonyok és a vállalati szereplők belépése következtében – már a megengedő licenctipus aránya növekszik. [144]

A GPL jellegű, kötött licenc előnye, hogy általa egyetlen piaci szereplő sem veheti magához a kódot, hogy

[30] End User License Agreement, magyarul Végfelhasználói licencszerződés, vagy másképp végfelhasználói engedély.
[31] Open Source Initiative, nyílt forrású licencek OSD (Open Source Definition) kompatibilitásának tanúsítást végző szervezet. https://opensource.org/

aztán saját üzleti változatot készítsen belőle. Például ha a Linux kernelhez egy vállalat saját üzleti modelljét támogatandó fejleszt valamit, az innováció előnyeit a teljes közösség élvezheti, ami gyorsabb fejlődést és magasabb termékminőséget eredményez mint amit egyetlen szereplő önmagában elérhetne [304]. A kötött licencek egyik hátránya viszont, hogy bennük foglalt "royalty-free" kitétel miatt, nem használhatnak bizonyos szabadalmakat [221].

A kialakult helyzetet árnyalja, hogy sok cég kettős licencelést alkalmaz, azaz termékük elérhető valamilyen nyílt forrású licenctípus alatt, ugyanakkor üzleti licencet is kínálnak azok számára akik a nyílt forrású licenc kötöttségeit nem tudják vagy akarják vállalni. További problémákat vet fel, ha az integrátor előre fizetett (upfront) licensz helyett időszakos (subscription-based) konstrukciót vesz igénybe. Ha ugyanis az integrátor saját termékében továbbértékesíti a FLOSS komponenst és később megszünteti az előfizetését, ügyfelei is elvesztik a jogot a komponens használatára. Ez a probléma a örökös (perpetual) licencelési konstrukcióval orvosolható, aminek meglétét érdemes ellenőrizni. [30]

Az állami szintű jogi szabályozás nem volt mindig teljesen tisztázott. Sokszor felmerült – bizonyos esetekben még ma is felmerül – a kérdés, hogy a jogalkotók engedélyezik-e vagy tiltják a FLOSS rendszerek integrációját, és ha igen milyen szinten [143]. Végül a kormányzati szférában megjelenő egyre nagyobb arányú FLOSS termék felvetette egy európai uniós nyílt forrású licenc szükségességét, amelynek első verziója 2007 januárjában meg is jelent EUPL[32] néven. Jelenleg a a 2017 májusában publikált EUPL v1.2-es verzió érvényes, amely ugyanazon év júliusában az OSI tanúsítást is megszerezte.

Gyakran felhozott érv, hogy FLOSS használata esetén nem tisztázott vagy nincs felelőség vállalás [162]. A nyílt licencekben szereplő "public" szó (lásd. GPL) hozzájárulhatott a téves elképzelés kialakulásához, miszerint az ilyen szoftver a "köz" tulajdona, azaz senkié, magyarán nincs senki, akit felelősségre lehetne vonni a hibás vagy kárt okozó működés esetén. E félreértés folytán a felelősségvállalás hiánya gyakran szerepel az ellenzők érvei közt. A helyzet az, hogy az üzleti gyártók a legritkább esetben vállalnak felelősséget a termékük által okozott kárért, azaz a felelősség kérdése ez esetben is ugyanúgy nyitott, másrészt a nyílt projektekben szerepet vállalók éppúgy felelősségre vonhatóak a szándékos károkozásért mint bárki más, legfeljebb azonosításuk lehet nehezebb (ami nem feltétlen igaz, lásd FS-F-M) [233]. A magas biztonsági követelményű rendszerek esetében – ahol emberéletek foroghatnak kockán – egyébként sem a felelősök

[32] European Free/Open Source Software licenc. https://joinup.ec.europa.eu/collection/eupl/introduction-eupl-licence#section-4

azonosíthatóságán, hanem a biztonsági esemény elkerülésén kell legyen a hangsúly. Nem árt tudni azt sem, hogy amennyiben az integrátor vagy fejlesztő FLOSS komponenseket használ fel – márpedig ez a hatás egyre erősödik – nem tehető felelőssé a FLOSS termék által okozott károkért [30].

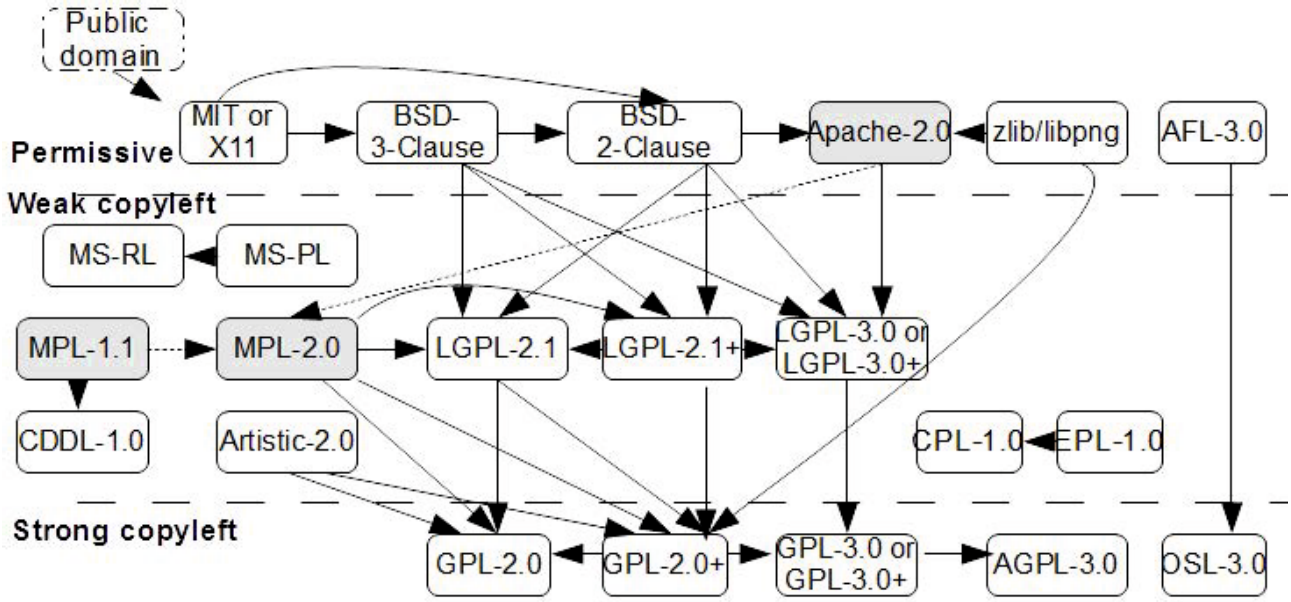
FLOSS termékfelhasználás során az összetett licencelés problémáinak kezelése kimerül a licencek nyilvántartásának szükségességében. Ha a termékeket nem módosítjuk és komponensként sem használjuk fel – azaz "dobozos" termékként használjuk fel – nincs szükségünk a licencekben biztosított jogok nagy részére, elegendő megbizonyosodni róla, hogy a szervezet megfelel-e az ott leírt követelményeknek. Sajnos néha még ez sem teljesen magától értetődő, amint azt Jaeger a nem egyértelműen megfogalmazott nyílt licencek (pl. CPOL) kapcsán megjegyzi [228]. Az egyszerű felhasználás azonban nem biztosítja a módosítás és integrálhatóság egyedi előnyeit, amelyek segíthetnek növelni a reakciókészségét, gazdaságosságát, rugalmasságát és a rendszer általános biztonsági szintjét, ezért a dobozos termékként való felhasználás gyakran nem elegendő.

A belső komponensfelhasználás során a licenc, copyright és IP jogok együtt határozzák meg a felhasználhatóság kritériumait, ez pedig komoly hatással lehet a rendszer szerkezetére [164, 375]. Következésképpen belső fejlesztések esetén a licencek kezelésének szabályozása nem megkerülhető. Az nyílt licencű termék felhasználása közvetett veszélyeket hordoz. A megkérdezett piaci szereplők általában az üzletfolytonosság kockázatát, a FLOSS közösség feletti irányítás képességének hiányát, a licencek összetettségét, értelmezési nehézségeket szokták említeni [229]. Nem elegendő tehát az integrálandó komponens API felületét, minőségét és programozási nyelvét vizsgálni, mindig figyelembe kell venni a licencelés kérdését is [208].

A FLOSS rendszerek összetett függőségeinek következményeként (FS-F-I) gyakran előfordul, hogy egyetlen kívánt komponens számos rejtett licenckötelmet tartalmaz. Ezek a rejtett licencek lehetnek más típusú esetleg kötöttebb nyílt forrású licencek, de zárt rendszerek licenckötelmei vagy IP jogai [376] amelyek ellentmondásba kerülhetnek egymással [105, 147, 377]. Például a Debian projekt csomagjainak közel 30%-ban nem lehetett semmilyen licencet automatikusan azonosítani[378] és a csomagok egy része inkonzisztens licencelést[33] alkalmaz[379, 380]. Az összetett licencrendszer nyilvántartása, a változások követése sőt a licencek puszta felderítése is komoly kihívást jelenthet [381]. Nem meglepő, hogy a projektek jelentős részében jelentkezik valamilyen szintű jogsértés (Kapitsaki eredményei szerint az esetek 60%-ában)[382].

[33] A licencelés inkozisztens lehet, ha a felhasznált (upstream) komponensek licencelése nem kompatibilis a downstream projekt licencelésével.

A nyílt forrású licencek függőségeinek összetettségét a 2.8. ábra demonstrálja.



2.8. ábra: Nyílt forrású licencek kompatibilitási függőségei. Forrás: [382]

A licencváltozás igazoltan élő jelenség. Di Penta népszerű FLOSS rendszereket célzó statisztikai elemzése során arra jutott, hogy a szoftverek felében az egyes fájlrevíziók során 60%-ban történt valamilyen változás, míg a másik fél esetében ez az érték 20-40% körül mozgott [164]. Jelenleg a licencváltások során a nyíltabb licencek felé történő elmozdulás figyelhető meg[383].

A szükséges de nem éppen egyszerű licencekezelési feladatot tovább nehezítik az esetleges licenchibák, a hibásan másolt vagy hiányzó licencek.

A licencekkel kapcsolatos (felhasználói) igények és feladatok vázlatosan az alábbiak:

- A rendszerben használt modulok azonosítása (forrásfájlok, fordítási kimenetek és könyvtárak);
- licencek azonosítása az állományokban;
- programmodulok függőségeinek feltárása;
- licenc kompatibilitási szabályok meghatározása;
- licencelési problémák azonosítása;
- licenc analízis prezentálása és vizualizációja.

Amennyiben nincs megfelelő erőforrásunk a licencelési feladatok belső kezelésére, a problémát licenckezelő rendszerek alkalmazásával orvosolhatjuk. Ilyen rendszer például az ODRL[384], ASLA, LIDESC, OSLC vagy a FLOSSology, amelyek automatikus licencanalízist, függőségelemző szolgáltatásokat nyúj-

tanak [326, 385]. Ezek a rendszerek egyszerű regular expression technikával (pl. OSLC) vagy a hagyományosan protein illesztéshez használt Binary Symbolic Alignment Matrix módszerével (FLOSSology) [386] azonosítják be a licenceket.

Létfontosságú Információs Rendszerelemek esetén ritkább, de nem kizárható az akvizíció esete. Ilyen esetben a szervezetnek a beszerzésre kerülő teljes IT struktúrát jogi megfelelőség szempontjából át kell vizsgálnia [75, 374] ami jelentős többletterhet jelenthet az összetett licenszelési struktúrát és rejtett buktatókat figyelembe véve[53, 387].

A felmérések tanúsága szerint a fejlesztők számára mind a mai napig (2018) nem világos melyik licenctípust kellene választaniuk az általuk favorizált célok elérése érdekében [387]. Pedig ez éppolyan fontos mind a nyílt komponenseket felhasználó zárt fejlesztések esetén, mind a nyílt közösség előnyeinek kihasználását célzó tevékenység során. A szükségtelen korlátozásokat tartalmazó licenc egyaránt elriaszthatja a potenciális fejlesztőket és felhasználókat [44, 388]. Ilyen esetekben általában az egyszerű, jól ismert licencmodellek választása vezet eredményre. Kimutatható, hogy a kevésbé kötött licencformák előnyösebbek ha a származtatott szoftver fejlesztéséhez komoly erőforrások szükségesek, míg az szigorúbb megkötéseket tartalmazó (copyleft) licencek az uralkodók ha az eredeti FLOSS és a származtatott szoftver létrehozása kisebb erőfeszítést igényelt[389]. A kötött licencek választása úgyszintén hátrányos ha a projekt a fejlesztőket célozza, míg előnyös, ha a felhasználókat vagy adminisztrátorokat [73]. A helyes licencválasztás tehát erős hatással lehet a projekt végső sikerességére.

Ez a folyamat is kétoldalú, ugyanis a közösség által felajánlott módosítások jogi státusza is hordozhat kockázatot, hiszen a módosítás szellemi termék, ügyelni kell tehát a hozzájárulások elfogadását megelőző formális szerződés meglétére [390]. Itt nem kell bonyolult jogi procedúrára gondolni, elegendő a fejlesztő rövid állásfoglalása, amely helyére teszi az általa készített szellemi termék jogi státuszát.

2.7.3 Tanúsítványok hiánya (FS-SZ-T)

A nyílt forrás lehetőségeinek kiaknázását erősen korlátozza, hogy a közösségi fejlesztési metódusból eredően nincs lehetőség az üzleti megoldások esetében szokásos tanúsítási eljárások (például Common Criteria) elvégzésére. Ennek a hiányosságnak súlyos következményei lehetnek, hiszen az olyan magas kockázattal

járó területeken – ilyen például az LRE – jogszabályi követelmény lehet valamilyen tanúsítás megléte[30] melynek hiányában a FLOSS termék felhasználását eleve ki kell zárni.

A probléma orvoslása kétféleképpen történhet. Vagy saját erőből, esetleg államilag finanszírozott keretek között kell elvégezni a tanúsítási eljárást, vagy olyan integrátor partnert kell keresni (amennyiben van ilyen) amely a kívánt komponens tanúsítását saját terméke keretében már elvégezte. Az integrátor közbeékelődésével viszont elveszítjük a módosíthatósággal járó előnyöket, hiszen a tanúsítás csak az adott összeállításra érvényes.

A probléma egy harmadik, formális módszeren alapuló figyelemre méltó megoldási lehetőségét vázolja fel Breuer és Pickin [391]. Eredményeik alapján létre lehet hozni olyan, központi szereplő nélküli tanúsítási eljárást, ahol bármely félnek lehetősége van ellenőrizni a tanúsítási eljárás bármely részét. Igaz, módszerükkel csak formálisan leírható kritériumok ellenőrizhetők, a vizsgálat tehát nem lenne teljes körű. Ennek ellenére mindenképpen figyelemre méltó alternatíva, ami sokat javíthatna a tanúsítási problémákkal küzdő FLOSS termékek megítélésén.

2.7.4 Kormányzati szabályozás (FS-SZ-K)

A legtöbb országban egyértelmű a politikai akarat a FLOSS termékek és nyílt szabványok bevezetését illetően, ez idáig azonban kevés valós lépés történt a szabályozás terén[12]. Sohn felhívja a figyelmet a FLOSS fejlesztők IP jogainak védelmére is, ami megkönnyítené és tisztázná a fejlesztők hozzájárulásainak jogállását[92].

Az Európai Unió létrehozta ugyan a korábban említett saját EUPL licencét, illetve több (idővel általában elhalt) projektet is indított a FLOSS bevezetését segítő minősítő rendszerek kifejlesztésére. Mindenesetre az átfogó szintű, netán nemzetközi szintű szabályozás egyelőre várat magára. Az Unió jelenlegi "Nyílt forrású szoftver stratégiája"[34] 10 elérendő célja között explicite szerepel a jogi kérdések tisztázása, az egyenlő feltételek biztosítása, az átláthatóság és jobb kommunikáció. Remélhetőleg ezek a célok mihamarabb megvalósulhatnak, mert a meglévő szabályozás sajnos sok esetben nem igazán illeszkedik a FLOSS-ra [392].

Akadnak persze kivételek. Az izlandi szabályozás például az elsők között foglalkozott a FLOSS rend-

[34] Open source software strategy: https://ec.europa.eu/info/departments/informatics/open-source-software-strategy_en

szerek fejlesztésének kérdéskörével [124], Kína, Norvégia, India, Izrael és Portugália szintén rendelkezik valamiféle FLOSS adaptációs szabályozással [59].

A hazai szabályozásban csak szórványosan, egyedi esetekre vonatkozó utalások szerepelnek, mint a a Kormányzati Adatközpont működéséről rendelkező 467/2017. (XII. 28.) Kormányrendelet, vagy az elektronikus közbeszerzés részletes szabályairól rendelkező 424/2017. (XII. 19.) Kormányrendelet. Bíztató jel viszont a E-közigazgatási Szabad Szoftver Kompetencia Központ 2012 végén történt megalakulása, melynek célja "a szabad szoftverek e-közigazgatási, illetve általánosságban intézményi és vállalati felhasználási lehetőségeinek vizsgálata és elősegítése."

A központ vállalt feladatai közé tartozik:

- egy szabad szoftver keretrendszer létrehozása amelybe "olyan szabad szoftverek kerülnek, amelyeket a Központ „ajánl" bizonyos feladatokra, illetve amelyek támogatása, honosítása megoldott és a minősítés szerint széles körben elterjeszthető."
- felhasználói és rendszer-üzemeltetési tanfolyamokat tart, és oktatási anyagokat készít;
- honosítási feladatokat lát el;
- "létrehoz egy mindenki számára szabadon elérhető tudástárat, amely egyrészt tartalmazza a Szabad Szoftver Kompetencia Központ működése során létrejövő esettanulmányokat, szoftverleírások, szakmai és oktatási segédanyagokat, cikkeket, stb. Másrészt gyűjtőhelyéül szolgál a különféle szabad licencek alatt elérhető, hazai és nemzetközi forrásból származó szabad szoftverekkel foglalkozó esettanulmányoknak, leírásoknak, szoftverdokumentációknak, könyveknek, cikkeknek, segédanyagoknak, stb."
- "fejlesztői révén megoldást kíván nyújtani a szabad szoftveres átállások során jelentkező hibákra, funkcionalitásbeli hiányosságokra, illetve informatikai fejlesztésekkel kívánja elősegíteni az interoperabilitás növelését".

2.7.5 Összefoglalás

Mint látható a FLOSS szabályozási kérdései meglehetősen összetettek. A licencelés sokszor nem triviális, a komponens alapú fejlesztés révén puszta felmérése is kihívás lehet. A szervezetek nem mindig vannak

tudatában a felhasználás pontos feltételeinek és nem ismerik a megfelelő licencű komponens vagy éppen a megfelelő licenctipus kiválasztásának helyes módszereit.

A korábban tárgyalt hiányosságok, elsősorban a tanúsítások hiánya miatt nagyon fontos lenne a magasabb szintű összefogás, hiszen a gyártók – ez esetben a közösségek – nem képesek megszerezni a szükséges tanúsításokat és nem is érdekeltek abban. A tanúsítás a felhasználó érdeke, amelyet viszont egyedileg rendkívül erőforrásigényes elvégezni, ami jelentősen hátráltatja a FLOSS termék kihasználhatóságát.

2.19. táblázat: A S kategóriában azonosított problémák

kód	szint	leírás	sajátosság
SS01	2	A megfelelő belső FLOSS szabályozás hiánya, átláthatatlan FLOSS felhasználáshoz vezet, a problémákat nem lehet megfelelően kezelni.	FS-SZ-B
SS02	2	FLOSS komponens-felhasználás esetén az összetett és rejtett licencelés a szervezet tudta nélkül kompromittálhatja a jogszabályi megfelelést, ami kikényszerített határidős módosításokhoz vezet, veszélyeztetve a rendszer rendelkezésre állását és integritását.	FS-SZ-L
SS03	2	A termék összetett függőségeiben ellentmondásos licencek találhatók vagy a licencelés megváltozik ami ellentmondáshoz vezet.	FS-SZ-L
SS04	2	FLOSS projektek ritkán szereznek tanúsítást. Következésképpen nem alkalmazhatók olyan területeken ahol a tanúsítvány megléte előkövetelmény.	FS-SZ-K
SS05	1	GPL licencű alkalmazások nem integrálhatják a szabadalomhoz kötött biztonsági megoldásokat	FS-SZ-L

2.20. táblázat: A S kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JS01	2	Dedikál FLOSS szabályzatot kell létrehozni vagy integrálni kell az egyedi elveket a saját eljárásokba	SS01
JS02	4	Azonosítani és követi kell a felhasznált FLOSS elemeket, azok licenceit és tisztázni a vonatkozó felelősségeketet és szerepeket.	SS01
JS03	3	A nyílt forrású licencek jogi keretet biztosítanak az azonnali helyi módosításokhoz, így azok szükség esetén külön megegyezés nélkül bármikor elvégezhetők.	

kód	szint	leírás	probléma
JS04	2	A fejlesztésben licenckezelő célszoftvereket kell használni.	SF10, SS02, SS03
JS05	1	A beszállítóktól meg kell követelni a licenc-tudatosságot, a szállított szoftver esetén a felhasznált komponensek licenclistáját.	SF10, SS02
JS06	1	Alternatív tanúsítási megoldások és/vagy központi költségvetésből finanszírozott hagyományos tanúsítási eljárások szükségesek.	SF05, SS04
JS07	1	A FLOSS kormányzati szintű szabályozása fontos lenne a FLOSS integráció jogi kérdéseinek tisztázása érdekében.	SS02, SS03, SS05

2.8 Terjesztés és támogatás (FS-T)

A nyílt modellből származó termékek terjesztése és a termékhez járó támogatás lehet teljesen megszokott vagy attól erősen eltérő. Amennyiben egy piaci szereplő átvállalja a terjesztést és a támogatást, a FLOSS felhasználó szempontból lényegében megkülönböztethetetlen a COTS termékektől. Amennyben viszont nincs ilyen szereplő vagy szolgáltatásait nem kívánjuk igénybe venni, a közösségi nyújtotta támogatás és a termék beszerzési módja jelentősen eltér a dobozos termékekétől.

Az eltérések a nyílt közösség szervezeti felépítéséből (FS-K-Sz), egyedi fejlesztési módszertanából (FS-F-P) és terjesztési gyakorlatából következnek és különösen a komponensek terjesztése esetében jelentkezik élesen.

A FLOSS könnyen elérhető, általában sűrű frissítési ütemezéssel rendelkezik ugyanakkor a közösség szinte egyáltalán nem fektet energiát a termék népszerűsítésébe. A alábbiakban ezekkel a jellemzőkkel foglalkozom.

2.8.1 Frissítések, kiadási ütemezés (FS-T-K)

A nyílt modell fejlesztői gyorsabb ki tudnak javítani egy-egy hibát [2] ám, ha az OSSD kiadási ciklusai túl sűrűek (FS-F-P) – előfordulhat, hogy havi 3-4 kiadás is van – az nem legkedvezőbb a gazdasági szereplőknek. Bár a hibajavítások sebessége szempontjából előnyös, a frissítés elvégzése idő és munkaigényes

[30]. Ebből kifolyólag előfordulhat, hogy a felhasználó vagy fejlesztő nem tölti le a frissítéseket [393]. Néhány FLOSS saját biztonsági frissítési rendszerrel rendelkezik (mint minden operációs rendszer), ám ha egy komponensről van szó akkor lehet, hogy a sérülékenység javításának a kiadás folyamatos követése az egyetlen módja. A probléma fordított irányban is fennállhat. Lehetséges, hogy a laza ütemezés folytán a FLOSS következő kiadása késik a felhasználó vagy integrátor ezért elkezdi a fejlesztői változatot használni[133]. A fejlesztői változat azonban többnyire nem garantálja a hibamentes működést, használata nyilvánvalóan nagyobb kockázattal jár.

A frissítések terjesztési módján kívül a nyílt és zárt modell között nincs hibajavítás terén eltérés, a hibajavítások sűrűsége és minősége a csapattól függ amely csinálja azt és nem a fejlesztési modelltől. Ha ugyanaz a csapat végez nyílt és zárt környezetben munkát a hibajavítás minősége is hasonló lesz [394, 395]. Az OSSD projektek javítási idejében viszont nagy eltérés tapasztalható, egyes projektekben (pl. AbiWord) akár éveken át javítatlanul maradhat hiba, míg mások napok alatt reagálnak [94].

A nyílt modell szociális jellegének hozadéka, hogy nem mindegy milyen módon igényeljük a hibajavítást. Lassenius arra jutott, hogy az udvariasan kért javításokat gyorsabban elvégzik mint az udvariatlanokat [396]. Ezt nehéz elképzelni az üzleti világban.

Az üzleti fejlesztésben tapasztalható határidő nyomás OSSD esetén nem jellemző[397]. Elvben lehetséges, hogy az üzleti fejlesztők a határidő nyomására kiadjanak olyan terméket, amelynek minőségellenőrzése még nem zajlott le vagy nem adott megfelelő eredményt [139]. Könnyen hozzáférhető adatok híján ezt azonban nagyon nehéz bizonyítani.

A fejlesztésről szóló fejezetben (FS-F-P) részletesen tárgyalom az extenzív újrahasznosításokból származó függőségek problémáját. A zárt rendszerek (pl. Microsoft Windows) általában úgy egyszerűsítik a függőségből eredő komplexitást, hogy megtartják a különféle verziókat ami egyrészt rendkívül redundánssá teszi a rendszert, másrészt megnehezíti a frissítést[94]. A nyílt módszertanra jellemző függő rendszerben viszont elegendő csak a hibás programkönyvtárat javítani és valamennyi függő rendszerben megszűnik a sérülékenység.

2.8.2 Terjesztés módja (FS-T-M)

A FLOSS termékek – különösen a szoftverek és programkönytárak – gyakran bináris formában is hozzáférhetők, de megkülönböztető jegy mindig az elérhető és újrafordítható forráskód. A forráskódban való terjesztés és fordítás átláthatóság szempontjából igen előnyös, ugyanis így garantálható, hogy a bináris valóban abból a forráskódból készült és nem tartalmaz nemkívánatos kiegészítőket. (Valójában ez csak bizonyos szintig igaz, hiszen a fordítókörnyezet, sőt maga a fordítóprogram is tartalmazhat sérülékenységeket vagy kiskapukat, de ettől a lehetőségtől itt most eltekintek.)

Kormányzati nyomásra néhány nagyobb gyártó (NDA[35] kötelezettség mellett) betekinthetővé teszi ugyan a forráskódját [398], de mivel a terjesztett forma bináris, nincs semmi garancia rá, hogy telepített kód valóban abból a forrásból készült. Ez a probléma egyébként FLOSS esetében éppúgy fennáll, amennyiben nem mi fordítottuk le a kódot, a disztribútoron múlik, hogy valóban azt kapjuk-e amit a forráskód állít [399].

Amennyiben egy bináris csomagot auditálunk, sajnos még a forráskód birtokában is nagyon nehéz hitelt érdemlően bizonyítani, hogy a bináris kód az adott forrásból készült. Egyrészt ismernünk kéne a fordításhoz használt valamennyi statikus könyvtár, fordítóprogram és segédprogram verzióját, másrészt tudnunk kéne a használt (optimalizációs és egyéb) fordítási kapcsolókat. Végül, ha minden egyéb azonos, fordítóprogram még akkor is bevihet némi nondeterminizmust a folyamatba időbélyegek, útvonalak vagy fordítási optimalizáció formájában.

Vannak kezdeményezések amelyek éppen ezt a problémakört igyekeznek orvosolni[400]. Ilyen például a Debian ReproducibleBuilds kezdeményezése, amely – ha egyszer működik – a teljes operációs rendszert bináris szinten tanúsíthatóvá teszi [401].

A bináris tanúsíthatósága fontos kritérium, hiszen a nyílt modell egyik legfőbb előnye – a forrás átláthatósága és ellenőrizhetősége – lényegében értelmét veszti ha nem tanúsítható bináris állományokat használunk.

Akár tanúsítható egy bináris akár nem, a bináris sértetlenségének biztosítása nagyon fontos. (Hacsak nem akarjuk elvégezni a tanúsítást minden egyes letöltésnél ami nagyjából értelmetlenné is teszi az egész bináris terjesztési modellt). A bináris letöltés alatti módosítása nagyon is valós támadási vektor [402]. Éppen

[35]Non Disclosure Agreement, jogi kötelezettségvállalás az átadott információ bizalmasságának megőrzésére.

ezért FLOSS binárisok sértetlenségét a legtöbb fejlesztő komolyan veszi és ellenőrző összegeket vagy digitális aláírást biztosítanak valamilyen megbízható(bb) platformon, melyek segítségével a sértetlenség ellenőrizhető.

PKI[36] híján az ellenőrzőösszeg vagy az aláírás sem nyújt igazán magas szintű biztonságot, de legnagyobb probléma, hogy a felhasználó gyakran egyáltalán nem is ellenőrzi azt. Néhány projektben – így a Linux disztribúciókban is – GPG[37] kulcsokkal aláírt ellenőrző-összeg listák biztosítják a csomagok sértetlenségét, amit az operációs rendszer automatikusan ellenőriz.

A csomagkezelőkön keresztül történő disztribúció a FLOSS egyik vezető terjesztési formája. Így közvetítik a bináris és forrás állományokat a már említett Linux disztribúciók, de így jutnak el a felhasználóhoz a népszerű programozási nyelvek komponensei is. A tbl. 2.21 néhány olyan programozási nyelvet mutat be, amelyeknél kifejezetten szorgalmazott és igen széleskörű a csomagkezelő használata.

2.21. táblázat: Néhány népszerű programozási nyelv és a hozzá tartozó csomagtárolók

programnyelv	tároló	csomagkezelő
perl	CPAN	ppm
python	Python Package Index (pyPi)	pip
Node-js	npm	npm
Haskell	Hackage	cabal
OCaml	OPAM	opam
ruby	RubyGems	gems
java	Central Maven	mvn

Ezek a csomagkezelők nagy mennyiségű adatot mozgatnak és bonyolult függőségeket kezelnek[403], használatuk biztonsági kockázatot jelenthet [404], s bár minden ilyen rendszer alkalmaz valamiféle sértetlenség védelmet, a csomagkezelő jó támadási vektor lehet, mivel a fejlesztők általában megbíznak benne.

[36] Public Key Infrastructure
[37] The GNU Privacy Guard, az egyik népszerű PGP implementáció

2.8.3 Egyszerű hozzáférhetőség (FS-T-E)

A FLOSS természeténél fogva nagyon könnyen elérhető. Nincs szükség vásárlásra, egyezmények elfogadására, nincs demó változat; a beszerzéshez mindössze egy hálózati kapcsolat kell és néhány kattintás. Éppen ez népszerűségének egyik forrása[108, 405] és az egyre növekvő mértékű FLOSS komponens felhasználás oka[406].

A könnyű hozzáférhetőség azonban biztonsági kockázatokat rejthet.

A nyílt projektek például jóval az elkészülésük előtt hozzáférhetőek lehetnek[407] ami a zárt forrás esetén a legritkább esetben fordul elő. Nem elég tehát kiválasztani a számunkra legmegfelelőbb változatot, annak készültségi állapotáról is meg kell győződni.

A FLOSS komponensek és gyakran a szabad szoftverek beszerzésének legelterjedtebb módja a keresőmotorral való keresés [208]. Az ilyen beszerzés komoly kockázatot hordoz, ugyanis a jelentős számú változat és a könnyű bináris készítési lehetőség folytán egyáltalán nem garantált, hogy a beszerző a megfelelő változatot találja meg a megfelelő terjesztőtől. A nyílt közösség marketing tevékenységet ritkán végez (FS-T-L) így SEO[38] technikákkal a keresőmotor viszonylag könnyen manipulálható a hibás vagy kiskaput tartalmazó módosított változat a lista elejére juttatható.

Az eredeti fejlesztői oldal beazonosítása egyáltalán nem egyértelmű, a GitHub projektek számtalan forkot tartalmaznak, a honlapok pedig nem rendelkeznek egyértelmű azonosító jegyekkel. Az elég széles körben használt és egyértelműen biztonság-kritikus PuTTY SSH kliens program eredeti fejlesztői oldalának címe például https://www.chiark.greenend.org.uk/~sgtatham/putty/. Könnyű elképzelni, hogy a fejlesztési környezetet nem ismerő felhasználó egy putty-dev.com vagy hasonló nevű forrást – helytelenül – sokkal hitelesebbnek vélne.

2.8.4 Lobbitevékenység hiánya (FS-T-L)

Több publikáció is eltérésként azonosította a lobbitevékenység hiányát [78]. Nem valószínű, hogy a terjesztéssel kapcsolatos de alapvetően gazdasági aspektusnak közvetlen biztonsági hatásai lennének, de a

[38] Search Engine Optimalization, weboldalak és azok kapcsolatainak optimalizálása a keresőtalálatban elért helyezések maximalizálása céljából. Egyes változatai (Black SEO) a kereső gyártója által tiltott vagy ellenjavalt módszereket is alkalmaznak.

közvetett hatások miatt érdekes lehet foglalkozni vele.

Mint láthattuk a jól megkülönböztethető márkajelzés és cégdizájn hiánya megnehezíti a hiteles változat beazonosítását. Az gyártók számára nagyon fontos, hogy használják a termékeiket így sokat is költenek rá[65], ezzel szemben a FLOSS használati mértéke általában indifferens az OSSD közösség számára, így még akkor se költenének rá, ha lenne erre fordítható forrásuk.

Az OSSD kevés figyelmet fordít a marketingtevékenységre. Nincsenek hírlevelek, nincs reklám, a honlapok erősen technikaiak, nehéz beazonosítani a felhasználók számára fontos előnyöket, következésképpen a felfogott érték kevesebb lehet mint zárt forrás esetében [408]. A marketing hiánya oda vezet, hogy a felhasználó végül zárt megoldást választja, még ha a FLOSS változat számára sokkal kedvezőbb is lett volna. A szervezetek vezetői magabiztosabbak azokkal a termékekkel kapcsolatban amelyek hatásának jobban ki vannak téve [78] függetlenül azok valós értékétől.

2.8.5 Támogató tevékenység eltérései (FS-T-T)

A FLOSS használata ellen felhozott egyik leggyakoribb érv a hivatalos támogatás hiánya[70, 98, 164, 238]. A gyártók szerződésben vállalnak kötelezettséget a frissítések és karbantartás elvégzésére, egy nyílt közösség nem nyújt ilyen garanciát [329]. Ez természetesen elsősorban a klasszikus nyílt fejlesztésekre vonatkozik, hiszen az irányított nyílt projektek gazdasági szereplőinek gyakran éppen a támogatás nyújtása jelenti a megélhetést (FS-G-G).

A klasszikus modell is nyújt valamilyen informális támogatást, ami világméretű, folyamatos és ingyenes [139] ám ez a támogatás sok tekintetben eltér a megszokottól:

- A felhasználónak esetleg sokat kell várnia, nincs szerződésben rögzített időkorlát a javításra [30], ha nincs központi frissítési lehetőség, a felhasználó felelős a frissítésekért [214]. A javítások támogatással gyorsíthatók, amit adott esetben be is lehet számolni a költségvetésbe [179]. A szponzorált nyílt közösségek a szponzorrációval szerzett pénzügyi forrásokat jórészt korrekciós és karbantartási folyamatokra költik hibák javítására kitűzött díjak vagy közvetlen fizetés formájában, ilyenformán az ilyen projektek hibajavító képessége sokkal jobb[134].
- a hibakeresés saját erőből is megoldható [123], de szakképzett munkaerőt igényel és bevezet egy

újabb kockázati tényezőt, hiszen a nehezen megszerzett szakértő elhagyhatja a céget [409].

- A segítségnyújtás alternatív csatornákon folyik Q&A portálok (pl. StackExchange), fórumok formájában ami más hozzáállást igényel[123]. Ezek átvehetik a helpdesk szerepét – gyakran nagyon hatékonyan – de nem megbízhatóak, nincs garantált megoldás és válaszidő [30, 141, 162, 410]
- általában lehetőség van támogatást vásárolni a fejlesztésben részt vállaló vagy kifejezetten ilyen szolgáltatást nyújtó cégektől [411] (pl. kettős licencelés (FS-SZ-L))o
- A FLOSS támogatottsága nem függ a piaci szereplő piaci sikerétől. Akár sikeres a termék akár nem, amíg fejlesztőket tud vonzani is támogatása biztosított[407]. A FLOSS projekt sem garantálja az üzletfolytonosságot[61] de ha az alapító cég ki is lép a partnerei folytathatják a fejlesztést[40]. A 10 évnél idősebb projektek megszűnésének esélye kicsi, ezek általában mindig tovább fejlődnek [351].
- Nincs képzés. Az oktatás autodidakta módon folyik dokumentációk, fórumok és listák segítségével.
- Nincs szervezett szakemberképzés. Az cégek összetettebb termék esetén odafigyelnek rá, hogy képzések által biztosítsák a szakembereket[143]. Az képzett szakembereket nehéz lehet megtalálni, tudásszintjük mérésére nincs formális lehetőség.
- ha van szolgáltatásként elérhető támogatás, az jobb minőségű lehet a kialakuló verseny miatt. A FLOSS támogatását bárki végezheti, senki nincs monopol helyzetben[108, 152].
- vannak területek (pl. beágyazott termékek piaca) ahol az üzleti támogatás gyenge, a FLOSS szakértők alkalmazása hozhat csak megoldást [244].

2.8.6 Összefoglalás

Amennyiben nem áll rendelkezésre üzleti támogatást vagy nem kívánjuk azt igénybe venni a nyílt modell terjesztési metódusai egyértelmű eltéréseket mutatnak, amelyek a biztonságra is hatással lehetnek.

A hiteles forrás és változat beazonosítása nem egyértelmű, lobbitevékenység és jogi támogatás hiányában a magukat hitelesnek mutató szereplők elleni fellépés gyenge vagy nem is létezik. A közösséget nem ismerő felhasználó hibás vagy félkész terméket tölthet le, illetve előfordulhat, hogy azonosítatlan forrásból más – esetleg kiskaput tartalmazó – változatot szereze be.

A közösségi támogatás megbízhatatlan, használata időben nehezen tervezhető, szociális kézségeket követel, ugyanakkor a piaci siker egyáltalán nem befolyásolja a támogatás minőségét. Amennyiben van piaci

alapon elérhető támogatás, a támogatást végző cégeket nem kötik engedélyek így a monopol helyzet hiánya és a folyamatos verseny folytán a szolgáltatás általában jobb minőségű.

A bináris alakban terjesztett termék vagy komponens sértetlensége elvben a forrás alapján tanúsítható, gyakorlatban viszont ez egyelőre nehézségekbe ütközik így a csomagolást végző fél iránti bizalom továbbra is követelmény. Mindamellett akár forrásról, akár binárisról legyen szó, a sértetlenséget biztosító kriptográfiai ellenőrzőösszegek vizsgálatát feltétlenül el kell végezni.

A belső támogatás alapját képező szervezett képzés gyakran hiányzik, így a megfelelő tudással rendelkező szakemberek felkutatása nehézségekbe ütközik. A formális képzési keretek (oklevél, tanusítvány) hiányában a szaktudás megléte is csak legfeljebb felvételi tesztekkel ellenőrizhető.

2.22. táblázat: A T kategóriában azonosított problémák

kód	szint	leírás	sajátosság
ST01	1	A gyors kiadási sűrűség és munkaigényes frissítés miatt a biztonsági javításokat nem vezetik át a downstream termékbe	FS-T-K
ST02	2	Nincs szerződésben biztosított időkorlát a javításra, a javítás késhet.	FS-T-K
ST03	2	Az udvariatlanul kért hibajavítást nem javítják megfelelő sebességgel.	FS-T-K
ST04	2	Kiadás csúszása miatt a fejlesztői verziót kezdik el használni.	FS-T-K
ST05	2	A szervezet félkész terméket tölt le.	FS-T-E
ST06	2	Nincs szakemberképzés, előfordulhat, hogy a szervezet nem talál megfelelő szakembert, nincs tanusítás a szaktudásra.	FS-T-T
ST07	2	Nincs szervezett képzés, az autodidakta tanulás minősége nem garantál, ami hibákhoz vezethet.	FS-T-T
ST08	2	A letöltött bináris változat nem a kiadott forrásból készült	FS-T-M
ST09	2	A letöltött bináris ellenőrzőösszegét vagy aláírását a felhasználó nem ellenőrzi.	FS-T-M
ST10	1	A keresőmotor nem a megfelelő verziót vagy disztribútort hozza fel.	FS-T-L
ST11	2	A felhasznált csomagkezelő rendszer (CPAN, hackage, deb) támadásával a felhasznált komponens támadható.	FS-T-M

2.23. táblázat: A T kategóriában azonosított javaslatok

kód	szint	leírás	probléma
JT01	2	Nyomon kell követni a frissítéseket.	SF10, SF13, ST01
JT02	2	Ellenőrzni kell a forráskód archívumának ellenőrző összegét, adott esetben a forráskódot	ST08, ST09, ST11
JT03	1	Meg kell határozni milyen forrásból és milyen feltételek mellett engedélyezett a kódfelhasználás.	SF10, SF13, ST04, ST08, ST10, ST11
JT04	3	Ellenőrizni kell a DVCS változáskészletét tanúsító digitális aláírásokat.	ST08
JT05	3	A FLOSS projektet az eredeti forrásból kell fordítani, a bináris integritásának biztosítása végett.	ST08, ST10
JT06	2	Össze kell gyűjteni és lehetőség szerint folyamatosan frissíteni a felhasznált FLOSS projekt sértetlenségének biztosítására használt valamennyi kriptográfiai nyílt kulcsot.	ST08, ST09
JT07	3	Kritikus esetben a hibajavítást önerőből kell és lehet megoldani.	SF07, SF11

2.9 Részkövetkeztetések

Megállapítottam, hogy a FLOSS fejlesztési módszertan és az általa létrehozott termék az üzleti modell esetében megszokottól eltérő, egyedi sajátosságokkal rendelkezik.

A KC-3 és KC-4 célok elérésének előkészítéseként kategorizáltam a lehetséges hatásokat és a kutatói közösség által lehetségesnek tartott ellenintézkedéseket.

Gazdasági és társadalmi hatás tekintetében nem tudtam egyértelmű biztonsági hatásokat azonosítani, azon túlmenően, hogy a jelenség ismerete előfeltétele a helyes védelmi eljárások kidolgozásának. Ugyanakkor az azonosított javaslatok közvetve, hosszabb távon befolyásolhatják a Létfontosságú Rendszerelemek biztonságát is. Ezek az intézkedések viszont csak globális szinten kivitelezhetők, általában meghaladják egyetlen szervezet lehetőségeit.

A fejlesztési modell, terméktulajdonságok, szociális struktúra, metaadatok és felhasználás kategóriák esetében több sajátosságot és lehetséges intézkedést is sikerült azonosítani. Ezeket részletesen a megfelelő

alfejezet végén tárgyalom.

A fentiek alapján megállapítom, hogy a nyílt fejlesztési modellben előállított FLOSS termékek olyan egyedi sajátosságokkal rendelkeznek, amelyek hatással lehetnek a FLOSS rendszert felhasználó LRE biztonságra tehát a H1 hipotézist elfogadtam.

Megállapítottam, hogy a FLOSS szabályozása jelenleg nem teljesen kiforrott. Jogalkotási tekintetben megindultak a kezdeti lépések, de vállalati szinten általában nincs kifejezett FLOSS specifikus szabályzat, esetleg egyáltalán nem is foglalkoznak a kérdéssel, az elfogadott tanúsítási rendszerek pedig nem illeszkednek jól a nyílt modellhez. A FLOSS licencelése összetett, és bár ez közvetlen biztonsági problémákat nem okoz, a rendelkezésre állást veszélyeztetheti.

3 Ellenintézkedések

A feltárt biztonsági kockázatok kezelésére a FLOSS felhasználó szervezetnek megfelelő védelmi intézkedésekre van szüksége. A kutatási céloknak megfelelően olyan átfogó, rendszerszintű és kockázatkezelési szempontokat is figyelembe vevő keretrendszert kerestem, amely alapján az egyes problémák kezelésére használható ellenintézkedéseket rendszerezett módon tudom összeszedni.

A lehetséges jelöltek a következő keretrenszerek voltak:

- A ISO 27001 szabvány, amely kevésbé technikai, inkább kockázatkezelési szemléletű, kis és nagyvállalatok számára is tartalmaz hat szakaszba gyűjtött javasolt módszereket. Publikusan közvetlenül nem elérhető.
- NIST Special Publication 800-53, kifejezetten biztonsági szemléletű dokumentum, amely 20 különféle adatvédelmi és biztonsági rendszabály csoportot definiálása segítségével segíti a USA szövetségi ügynökségeinek kockázatkezelését. Publikusan elérhető.
- A PCI DSS szabvány 12 javasolt módszert nyújt az adatbiztonság és biztonsági szabályozás kialakításához. Az elektronikus fizetőeszközök feldolgozására és tárolására koncentrál. Publikusan nem elérhető.
- COBIT/ITIL amely a ISO 27001-el szemben inkább ernyő keretrenszer (COBIT) illetve az IT szolgáltatások üzleti célokhoz illesztésével fókuszál (ITIL). Nem biztonság az elsődleges céljuk. ITIL process Security Management az ISO/IEC27001-en alapszik. Publikusan nem elérhető [412].
- NIST Cybersecurity Framework [413], kibertámadások megelőzésére, felderítésére és válaszlépésekre koncentráló szabályozási keretrendszer. Publikusan elérhető.

A szóba jövő lehetőségek közül a NIST Cyber Security Framework [22] v1.1-es verziója által használt rendszerezést használtam. A keretrendszer választását indokolja, hogy szabadon elérhető, kiforrott, rend-

szerelvű, kockázat orientált és nem utolsó sorban létfontosságú rendszerek igényeit is figyelembe vevő publikációról van szó amely eredetileg kifejezetten létfontosságú rendszerelemek kiberbiztonsági folyamatainak támogatására készült. A keretrendszer az implementációs szinttől a üzleti folyamatokon keresztül a vezetői szinten eldőlő kockázatkezelési döntésekig valamennyi területet összefogja, így megfelel a kutatás célkitűzéseiben megfogalmazott rendszerelvűség követelményének.

A NIST Cyber Security Framework Core öt nagy feladat alá sorolja be a kibervédelem kategóriáit:

- Identify azaz Azonosítás (ID);
- Protect azaz Védelem (PR);
- Detect azaz Felderítés (DE);
- Respond azaz Válaszlépések (RS);
- Recover azaz Helyreállítás (RC).

A keretrendszer a feladatokhoz összes 23 kategóriát definiál, amelyeket aztán további 108 alkategóriára bont fel. A kategóriák valamennyi nagyobb védelmi funkciót lefedik az eszközkezeléstől kezdve a kockázatkezelésen, fizikai védelmen, képzésen, adatvédelmen és sok más egyében át a helyreállítási és fejlesztési feladatokig. A keretrendszer nagy előnye, hogy hivatkozásokat tartalmaz a népszerű ajánlásokhoz, azaz minden egyes alkategóriánál pontos referenciákat találunk a CIS CSC, COBIT 5, ISA 62443, ISO/IEC 27001:2013 és a NIST SP 800-53 (Rev. 4) megfelelő szakaszaihoz. Ezeknek a hivatkozásoknak a segítségével könnyen beazonosítható a kontextus és a kategória esetében releváns védelmi intézkedések.

A FLOSS biztonsági sajátosságainak elemzése során meghatároztam és a valószínűsíthető problémákat és a publikációkban javasolt lehetséges ellenintézkedéseket, ennek megfelelően a fejezetben bemutatott ellenintézkedések meghatározását két forrás alapján végeztem az alábbiak szerint:

1. A korábban mások által javasolt ellenintézkedéseket beillesztettem a biztonsági keretrendszerbe ahol ezt indokoltnak találtam.
2. A keretrendszer által javasolt biztonsági intézkedések leírása alapján javasoltam valamilyen megoldást, amivel az adott FLOSS specifikus problémát kezelni lehet.

Minthogy a keretrendszer minden ponthoz meghatározza a vonatkozó biztonsági ajánlások megfelelő fejezeteit is, azok áttanulmányozásával véleményem szerint kellő pontossággal meg lehet határozni a probléma

rendszer szintű hatáspontjait és a kezeléséhez szükséges lépéseket.

Természetesen nem állt szándékomban teljes körű szabályzatot alkotni – ez az értekezés kereteit és egyetlen szerző lehetőségeit messze meg is haladná – a célom az volt, hogy szisztematikus módszerekkel be tudjam azonosítani mindazokat a hatáspontokat ahol a feltérképezett problémák ellen védekezni kell, ezzel segítve a szabályzat tervezők és döntéshozók későbbi munkáját.

3.1 Azonosítás (ID)

3.1.1 Eszközkezelés (ID.AM)

A szervezet üzleti céljainak eléréséhez szükséges adatokat, személyeket, eszközöket, rendszereket és létesítményeket be kell azonosítani és a szervezet céljainak és kockázatkezelési stratégiájának megfelelő relatív prioritással kezelni.

VID01	A szervezet meghatározza mely hardver elemek épülnek FLOSS alapokra.
Keretrendszer:	ID.AM-1 (COBIT 5 BAI09.01, BAI09.02,NIST SP 800-53 Rev. 4 CM-8, PM-5)
Vonatkozó problémák:	SH04, SH05, SM07
Szabályozás:	3.1.1.1.

Indoklás: A FLOSS elemeket tartalmazó rendszerek gyártóitól meg kell követelni a megfelelő FLOSS specifikus belső szabályozást. Ehhez ismerni kell melyek ezek az eszközök.

VID02	A szervezet meghatározza és katalogizálja milyen FLOSS rendszereket használ.
Keretrendszer:	ID.AM-2 (COBIT 5 BAI09.01, BAI09.02, BAI09.05,NIST SP 800-53 Rev. 4 CM-8, PM-5)
Vonatkozó problémák:	SG01, SH04, SH05, SH08, SM07, SS01, ST04, ST05
Javaslatok:	JS02
Szabályozás:	3.1.1.1.

Indoklás: A FLOSS terméket beszállítók és közösségek ellenőrzéséhez és minősítéséhez ismerni kell mely szoftverek készültek nyílt modell alapján.

VID03	A szervezet katalogizálja a felhasznált FLOSS rendszerkomponenseket.
Keretrendszer:	ID.AM-2 (COBIT 5 BAI09.01, BAI09.02, BAI09.05,NIST SP 800-53 Rev. 4 CM-8, PM-5)
Vonatkozó problémák:	SG01, SH04, SH05, SH06, SH08, SH10, SM07, ST04, ST05
Javaslatok:	JS02
Szabályozás:	3.1.1.1.

Indoklás: A szervezet által közvetve vagy közvetlenül felhasznált FLOSS komponensek csak úgy elemezhetők megfelelően ha helyük és szerepük a szervezet információs rendszereiben ismert.

VID04	A szervezet meghatározza mely FLOSS elemeket veszi igénybe beszállítón keresztül és melyeket használja közvetlen módon.
Keretrendszer:	ID.AM-2 (COBIT 5 BAI09.01, BAI09.02, BAI09.05,NIST SP 800-53 Rev. 4 CM-8, PM-5)
Vonatkozó problémák:	SF02
Javaslatok:	JS02
Szabályozás:	3.1.1.1.

Indoklás: A beszállítótól származó FLOSS elemek kezelése hasonlít a COTS szoftverekéhez, de szükség lehet egyedi eljárásokra. A közösségi forrásból felhasznált FLOSS eljárásai a megszokottól nagy mértékben eltérhetnek. Emiatt nem elegendő a szoftverplatform vagy alkalmazás típusát megállapítani, pontosan ismerni a beszerzés jellemző módját is.

VID05	A szervezet felméri, milyen FLOSS komponensek tárolóját követi és mely fejlesztésekben vesz részt.
Keretrendszer:	ID.AM-3 (COBIT 5 DSS05.02,NIST SP 800-53 Rev. 4 AC-4, CA-3, CA-9, PL-8)

Vonatkozó problémák:	SF06, SF14, SG01, SH03, SH04, SH05, SH06, SH09, SK06
Javaslatok:	JF01
Szabályozás:	3.1.1.1.

Indoklás: A FLOSS komponensek folyamatos fejlesztés igényelnek, a fejlesztést nyomon kell követni, a módosításokat lehetőség szerint vissza kell vezetni az eredeti projektbe. A komponensmódosítások visszavezetése során információ szivároghat ki.

VID06	A szervezet felméri milyen FLOSS szoftvertárolókat használ, ide értve az operációs rendszer által használt, programozási nyelv specifikus és egyedi szoftvertárolókat is.
Keretrendszer:	ID.AM-4 (COBIT 5 APO02.02, APO10.04, DSS01.02,NIST SP 800-53 Rev. 4 AC-20, SA-9)
Vonatkozó problémák:	SG01, SH03, SH04, SH05, SH06, SH09, ST11
Javaslatok:	JF01
Szabályozás:	3.1.1.1.

Indoklás: A szoftvertárolókon keresztül információ áramlik a szervezetbe. Biztonsági riasztás esetén rövid időn belül el kell tudni dönteni, hogy a szervezet érintette-e vagy sem.

VID07	A szervezet meghatározza és kommunikálja a beszállítóitól elvárt felelősség szintjét a nyílt forrású komponensekkel kapcsolatban.
Keretrendszer:	ID.AM-6 (COBIT 5 APO01.02, APO07.06, APO13.01, DSS06.03,NIST SP 800-53 Rev. 4 CP-2, PS-7, PM-11)
Vonatkozó problémák:	SF05, SF11, SH01, SH02, SH05
Szabályozás:	3.1.1.1.

Indoklás: A FLOSS elemeket felhasználó partnerek nem feltétlen tehetők felelőssé a harmadik féltől származó komponensekből eredő hibákért. A felelősség kérdése tisztázásra szorul. Ellenőrizni kell, hogy

a beszállító ismeri és alkalmazza a nyílt forrású elemekkel kapcsolatos alternatív minőségértékelési módszereket, a fejlesztés során az egyedi kihívásoknak megfelelő minőségbiztosítási metódust alkalmaz.

VID08	A szervezet meghatározza a FLOSS felhasználással kapcsolatos felelősségi köröket és személyeket.
Keretrendszer:	ID.AM-6 (COBIT 5 APO01.02, APO07.06, APO13.01, DSS06.03,NIST SP 800-53 Rev. 4 CP-2, PS-7, PM-11)
Vonatkozó problémák:	SG01, SH04
Szabályozás:	3.1.1.1., 3.1.3.3.

Indoklás: A FLOSS biztonsági hatásainak kezelése szervezett formát kíván. Ha nincs felelős és jól definiált szabályozás, elképzelhető, hogy a szervezet rejtett módon használ fel FLOSS komponenseket. A COTS esetén bevált módszertan a FLOSS komponensek minősítésére nem megfelelő. Olyan felelős személyt kell kijelölni, aki megfelelő jártassággal (esetleg képesítéssel) rendelkezik a FLOSS rendszerek kiválasztása, minősítése, integrálása és üzemeltetése terén.

3.1.2 Kormányzás (ID.GV)

A szervezet szabályozási, jogi, kockázati, környezeti és műveleti követelményeinek kezeléséhez és nyomon követéséhez szükséges szabályok és eljárások egyértelműek és megfelelő információval látják el a kiberbiztonsági kockázatmenedzsmentet.

VID09	A szervezet informatikai biztonsági szabályzatában foglalkozik a FLOSS kérdésével.
Keretrendszer:	ID.GV-1 (COBIT 5 APO01.03, APO13.01, EDM01.01, EDM01.02,NIST SP 800-53 Rev. 4 -1 controls from all security control families)
Vonatkozó problémák:	SF05, SS01
Javaslatok:	JS01
Szabályozás:	3.1.1.1.

Indoklás: Az üzleti termékek kezelésére használt szabályzatok a FLOSS komponensek esetében nem kielégítőek. A FLOSS egyedi sajátosságainak figyelembe vevő szabályzatot kell létrehozni vagy a meglévő szabályzatot kiegészíteni. Ennek hiányában az átláthatatlan FLOSS felhasználás várhatóan negatívan befolyásolja a biztonságot és jelentősen megnehezíti a kockázatbecslést.

VID10	A szervezet kijelöli a FLOSS felhasználásért felelős személyt és meghatározza az igénybe vehető külső partnereket és azok szerepét.
Keretrendszer:	ID.GV-2 (COBIT 5 APO01.02, APO10.03, APO13.02, DSS05.04,NIST SP 800-53 Rev. 4 PS-7, PM-1, PM-2)
Vonatkozó problémák:	ST06, ST07
Szabályozás:	3.1.1.2.

Indoklás: A FLOSS kezelése kapcsán nem minden feladat oldható meg önerőből. Tovább nehezíti a helyzetet a nehezen beszerezhető szakképzett munkaerő és a FLOSS irányultságú szervezett képzés hiánya. A probléma súlyossága külső partnerek tudásának igénybevételével mérsékelhető, amely esetben szükség van a külső partnerekkel való kapcsolattartás szabályozása és a felelősségek meghatározása.

VID11	A szervezet meghatározza milyen forrásokból és partnerektől származó forráskód használható a különféle biztonsági besorolású alrendszereiben és azok használatát milyen szabályokhoz köti.
Keretrendszer:	ID.GV-2 (COBIT 5 APO01.02, APO10.03, APO13.02, DSS05.04,NIST SP 800-53 Rev. 4 PS-7, PM-1, PM-2)
Vonatkozó problémák:	ST08, ST10, ST11
Javaslatok:	JT03
Szabályozás:	3.1.3.1., 3.3.11.4.

Indoklás: A FLOSS nagyon könnyen hozzáférhető, ugyanakkor az ellenőrizetlen felhasználása súlyos kockázatot jelent. A megbízható forrás beazonosítása nem egyértelmű feladat, a felhasznált forrás direkt vagy indirekt módon további forrásokat használhat fel. A kockázatvállalás szintje az egyes alrendszerekben eltérő lehet ezért célszerű területenként külön, világosan megfogalmazni a FLOSS komponensek és

szoftverek elfogadható beszerzési forrásait. Források alatt értendőek a partnerek oldalai, letöltési portálok, szoftvertárolók, a programnyelvek szoftvertárolói és a segítségnyújtó oldalakon fellelhető forráskód részletek is.

VID12	A FLOSS felhasználásért felelős személy ellenőrzi, hogy a felhasznált FLOSS elemek kielégítik-e a kiberbiztonság vonatkozó jogi követelményeit.
Keretrendszer:	ID.GV-3 (COBIT 5 BAI02.01, MEA03.01, MEA03.04,NIST SP 800-53 Rev. 4 -1 controls from all security control families)
Vonatkozó problémák:	SS04, SS05
Szabályozás:	3.1.1.2., 3.1.2.3., 3.1.4.2.

Indoklás: A jelenlegi jogszabályok nem tartalmaznak FLOSS specifikus kitételeket, így más rendszerekkel azonos követelmények vonatkoznak rájuk, ugyanakkor az egyedi sajátosságok miatt nem feltétlenül tudják teljesíteni az előírt követelményeket. Ha például a jogszabály megnevezi a hiteles PKI tanúsítókat, ám ez a lista a FLOSS rendszerekben használt általában GPG alapú kulcsokat nem tartalmazza – ahogy jelenleg nem tartalmazza – a frissítéskezelés jogi háttere problematikussá válhat.

3.1.3 Kockázatfelmérés (ID.RA)

A szervezet számára világosak a szervezeti műveletekkel (küldetést, funkciókat és reputációt is ide értve), eszközökkel és személyekkel kapcsolatos kiberbiztonsági kockázatok.

VID13	A felhasznált FLOSS szoftverek és komponensek kódját és metaadatait biztonság-centrikus megbízhatósági modellekkel kell elemezni.
Keretrendszer:	ID.RA-1 (COBIT 5 APO12.01, APO12.02, APO12.03, APO12.04, DSS05.01, DSS05.02,NIST SP 800-53 Rev. 4 CA-2, CA-7, CA-8, RA-3, RA-5, SA-5, SA-11, SI-2, SI-4, SI-5)
Vonatkozó problémák:	SF03, SH01, SH02, SJ01, SJ02, SS04
Javaslatok:	JH01, JK01, JS06
Szabályozás:	3.1.2.1., 3.1.2.2.

Indoklás: FLOSS esetében általában nincs formális tanúsításra lehetőség a kockázatbecslést egyedi módszerrel kell elvégezni. A metaadatok (forráskód, hibajegyek, fejlesztői kommunikáció) analízisével pontosabb kép alkotható a projekt várható sérülékenységeiről. Ilyen elemzést tesz lehetővé például a CodeTrust.

VID14	A FLOSS várható sérülékenységeit a forráskód átnézésével becsülni lehet.
Keretrendszer:	ID.RA-1 (COBIT 5 APO12.01, APO12.02, APO12.03, APO12.04, DSS05.01, DSS05.02,NIST SP 800-53 Rev. 4 CA-2, CA-7, CA-8, RA-3, RA-5, SA-5, SA-11, SI-2, SI-4, SI-5)
Vonatkozó problémák:	SF03
Javaslatok:	JM12
Szabályozás:	3.1.2.1.

Indoklás: Specifikus célszoftverek híján a kódminőség a forráskód átnézésével is megbecsülhető.

VID15	A FLOSS eszközök sérülékenységeinek meghatározása érdekében a szervezet folyamatosan monitorozza a fejlesztő közösség metaadatait, vagy ilyen szolgáltatást nyújtó harmadik fél támogatását veszi igénybe.
Keretrendszer:	ID.RA-2 (COBIT 5 BAI08.01,NIST SP 800-53 Rev. 4 SI-5, PM-15, PM-16)
Vonatkozó problémák:	SM04
Javaslatok:	JK01, JK02
Szabályozás:	3.1.2.3., 3.3.3.2.

Indoklás: A FLOSS sérülékenységeit csak úgy lehet időben feltárni, ha a rendelkezésre álló valamennyi információt felhasználjuk. A támadó fél hozzáfér a forráskódhoz, hibalistákhoz és fejlesztői kommunikációhoz, amelyek alapján sérülékenységeket határozhat meg. A védekezés csak akkor lehet sikeres, ha ezeket az információkat a szervezet is felhasználja.

VID16	A fenyegetések meghatározása során figyelembe kell venni a FLOSS fejlesztés sajátosságait
Keretrendszer:	ID.RA-3 (COBIT 5 APO12.01, APO12.02, APO12.03, APO12.04,NIST SP 800-53 Rev. 4 RA-3, SI-5, PM-12, PM-16)

Vonatkozó problémák:	SF04, SF06, SF07, SF10, SF11, SF12, SF13, SF14, SM01, SM02, SM03, SM04, SM05, SM06, SM07, SM08, SM09, ST01, ST02, ST03, ST04, ST05
Szabályozás:	3.1.2.1.

Indoklás: A nyílt modell e kutatás során feltárt sajátos fenyegetéseit is figyelembe kell venni a kockázatmeghatározás során. Ilyen a szoftvertároló támadása, lassú javítás, közösség stabilitása stb.

VID17	A nyílt forrás egyedi jellegzetességeinek üzleti hatását és azok valószínűségét is figyelembe kell venni.
Keretrendszer:	ID.RA-4 (COBIT 5 DSS04.02,NIST SP 800-53 Rev. 4 RA-2, RA-3, SA-14, PM-9, PM-11)
Vonatkozó problémák:	SF02, SS01, SS03, ST02
Javaslatok:	JF03, JH05
Szabályozás:	3.1.2.1.

Indoklás: A nyílt forrás nem piacvezértelt, egyáltalán nem vagy nehezen befolyásolható pénzügyi eszközökkel, licencelési problémák illetve a forkolás veszélyeztetheti az üzemfolytonosságot.

3.1.4 Kockázatkezelési stratégia (ID.RM)

A szervezet a prioritásokat, kereteket, kockázat-határértékeket és feltevéseket meghatározta és használja működési kockázattal kapcsolatos döntések esetén.

VID18	A kockázatelemzés során a szervezet dedikált FLOSS kockázatelemző keretrendszereket alkalmaz a FLOSS szoftverei és komponenseinek elemzésere.
Keretrendszer:	ID.RM-1 (COBIT 5 APO12.04, APO12.05, APO13.02, BAI02.03, BAI04.02 ,NIST SP 800-53 Rev. 4 PM-9)
Vonatkozó problémák:	SF01, SF03, SH04
Javaslatok:	JJ01, JM09
Szabályozás:	3.1.2.1.

Indoklás: Az üzleti termékek kockázatelemzésére használt modellek nem illeszkednek jól a FLOSS rendszerekre. Egyedi modellek alkalmazása szükséges a belső és külső fenyegetések pontos meghatározásához.

VID19	A kockázatelemzés során a szervezet kvantitatív vizsgálatokat végez a FLOSS elemet jellemző adatokat dokumentálja.
Keretrendszer:	ID.RM-1 (COBIT 5 APO12.04, APO12.05, APO13.02, BAI02.03, BAI04.02 ,NIST SP 800-53 Rev. 4 PM-9)
Vonatkozó problémák:	SM11
Javaslatok:	JF03
Szabályozás:	3.1.2.1.

Indoklás: A FLOSS fejlesztőközösségének metaadatai segítségével kvantitatív adatokkal jellemezhető a termék minősége és a használatával járó kockázat. A mérőszámok összehasonlíthatóak, objektív elemzést tesznek lehetővé, ellentétben a zárt forrású komponensekkel, ahol a formális minősítés, a bizalom szintje vagy a felhasználói visszajelzések adhat támpontot a kockázatok megítélésénél. A megszerzett minősítés csak egy jól meghatározott környezetre és konfigurációra érvényes, a mérőszámok általánosabb és átfogóbb képet nyújtanak a projekt állapotáról.

VID20	A fejlesztést végző részleg vagy beszállító dedikált FLOSS komponens kezelő célszoftvert alkalmaz a kockázatbecslésekhez.
Keretrendszer:	ID.RM-1 (COBIT 5 APO12.04, APO12.05, APO13.02, BAI02.03, BAI04.02 ,NIST SP 800-53 Rev. 4 PM-9)
Vonatkozó problémák:	SF04, SH03, SH06, SS02, SS03
Javaslatok:	JH08
Szabályozás:	3.1.3.3.

Indoklás: A komponens kiválasztás folyamata összetett, a függőségek jelentősen megnehezítik illetve ellehetetlenítik a manuális feldolgozást. A komponens-kezelő célszoftver segít felderíteni a jogi kockázatokat és a sérülékenységek és függőségek okozta kockázatokat.

3.1.5 Beszállítói lánc kockázatkezelése (ID.SC)

A szervezet meghatározta és felhasználja a prioritásait, korlátait, kockázattűrő képességét és elképzeléseit a beszállítói lánc kockázataival kapcsolatos kockázati döntések során. A szervezet összeállította és megvalósította a beszállítói lánc kockázatának felméréséhez és kezeléséhez szükséges folyamatokat.

VID21	A szervezet FLOSS specifikus SCRM folyamatokat is alkalmaz.
Keretrendszer:	ID.SC-1 (COBIT 5 APO10.01, APO10.04, APO12.04, APO12.05, APO13.02, BAI01.03, BAI02.03, BAI04.02,NIST SP 800-53 Rev. 4 SA-9, SA-12, PM-9)
Szabályozás:	3.1.2.1., 3.1.2.3.

Indoklás: A közösségek állapota, életpályája befolyással lehet a felhasznált termékkel kapcsolatos kockázatokra, ugyanakkor a hagyományos SCRM módszerek a közösségekkel kapcsolatban nem adnak megbízható eredményt.

VID22	Ha a szervezet FLOSS komponenseket használ fel, az azokat létrehozó közösségek életciklusát és állapotát is figyelembe veszi az SCRM folyamatok során.
Keretrendszer:	ID.SC-2 (COBIT 5 APO10.01, APO10.02, APO10.04, APO10.05, APO12.01, APO12.02, APO12.03, APO12.04, APO12.05, APO12.06, APO13.02, BAI02.03,NIST SP 800-53 Rev. 4 RA-2, RA-3, SA-12, SA-14, SA-15, PM-9)
Javaslatok:	JM11
Szabályozás:	3.1.2.3., 3.3.3.2., 3.3.7.2.2.

Indoklás: A FLOSS komponensek biztonsági frissítései erősen függenek a létrehozó fejlesztőközösség állapotától és működőképességétől.

VID23	A szervezet meghatározza mely beszállítói építenek FLOSS forrásokra vagy használnak FLOSS komponenseket. Felméri és nyilvántartja és minősíti a beszállítók által alkalmazott szabályzatot.
Keretrendszer:	ID.SC-2 (COBIT 5 APO10.01, APO10.02, APO10.04, APO10.05, APO12.01, APO12.02, APO12.03, APO12.04, APO12.05, APO12.06, APO13.02, BAI02.03,NIST SP 800-53 Rev. 4 RA-2, RA-3, SA-12, SA-14, SA-15, PM-9)
Szabályozás:	3.1.3.3., 3.1.3.3.2., 3.1.3.6.

Indoklás: Amennyiben a szervezet a felelősséget át kívánja hárítani a beszállítóira, meg kell bizonyosodni afelől, hogy a beszállítók tisztában vannak a FLOSS felhasználásából származó kockázatoknak és megfelelő eljárásokat implementáltak a kockázat kezelésére.

VID24	A szervezet támogatást nyújtó partnereket keres a felhasznált FLOSS komponensekhez.
Keretrendszer:	ID.SC-3 (COBIT 5 APO10.01, APO10.02, APO10.03, APO10.04, APO10.05,NIST SP 800-53 Rev. 4 SA-9, SA-11, SA-12, PM-9)
Vonatkozó problémák:	SK05
Szabályozás:	3.1.3.3.

Indoklás: Amennyiben van támogatást szolgáltatásként nyújtó vállalkozás, a szervezet visszavezetheti a kockázatkezelést a COTS termékeknél alkalmazott metódusokra.

VID25	A szervezet saját fejlesztőit integrálja a fejlesztőközösségbe, vagy támogatásokon keresztül biztosítja magának a szükséges irányítást.
Keretrendszer:	ID.SC-3 (COBIT 5 APO10.01, APO10.02, APO10.03, APO10.04, APO10.05,NIST SP 800-53 Rev. 4 SA-9, SA-11, SA-12, PM-9)
Vonatkozó problémák:	SH07, SK05
Javaslatok:	JK01, JK05, JK06
Szabályozás:	3.1.3.5., 3.1.3.8.

Indoklás: Üzleti partner hiányában nincs lehetőség szerződéskötésre. A közösség által fejlesztett termékkel kapcsolatos biztosítékokat csak a közösség játékszabályai alapján lehet megszerezni. A szervezetnek megfelelő lobbierőt kell biztosítania céljai eléréséhez. Ezt támogatás formájában és kapcsolatépítéssel érheti el. A személyes találkozások igazolható módon növelik a közösséggel való jó együttműködés esélyét.

VID26	A szervezet rendszeresen ellenőrzi a FLOSS komponenseket tartalmazó eszközeinek beszállítóit, hogy teljesítik-e az elvárt követelményeket.

Keretrendszer:	ID.SC-4 (COBIT 5 APO10.01, APO10.03, APO10.04, APO10.05, MEA01.01, MEA01.02, MEA01.03, MEA01.04, MEA01.05 ,NIST SP 800-53 Rev. 4 AU-2, AU-6, AU-12, AU-16, PS-7, SA-9, SA-12)
Vonatkozó problémák:	SK03, SK04, SM08
Szabályozás:	3.1.3.3.4., 3.1.3.8.

Indoklás: A nyílt fejlesztőközösség dinamikusan változik, a harmadik fél korábban fennálló kapcsolata megszűnhet vagy megváltozhat a közösséggel.

VID27	A szervezet folyamatos teszteket végez, melyek segítségével előrejelezhető a fejlesztő közösségben beálló változások mértéke és időpontja.
Keretrendszer:	ID.SC-4 (COBIT 5 APO10.01, APO10.03, APO10.04, APO10.05, MEA01.01, MEA01.02, MEA01.03, MEA01.04, MEA01.05 ,NIST SP 800-53 Rev. 4 AU-2, AU-6, AU-12, AU-16, PS-7, SA-9, SA-12)
Vonatkozó problémák:	SK03, SK04, SK05, SM02
Javaslatok:	JK04, JM11
Szabályozás:	3.1.3.8.

Indoklás: A nyílt közösség felbomolhat, megváltozhat, újraalakulhat. Ezeket a változásokat az SCRM folyamatnak időben időben észlelnie és kezelnie kell.

VID28	A szervezet ellenőrzi, hogy a hibajavítások reakcióideje és minősége az elvárható tartományba esik-e.
Keretrendszer:	ID.SC-5 (COBIT 5 DSS04.04,NIST SP 800-53 Rev. 4 CP-2, CP-4, IR-3, IR-4, IR-6, IR-8, IR-9)
Szabályozás:	3.1.2.1., 3.1.3.8., 3.1.5.6.

Indoklás: Amennyiben a közösség végzi a FLOSS támogatását, ez a támogatás nem fogalmazható meg elvárásként. Minőségének folyamatos ellenőrzése szükséges.

VID29	A szervezet felkészül a FLOSS esetleges forkolására és elemzi az ezzel járó feladatokat.
Keretrendszer:	ID.SC-5 (COBIT 5 DSS04.04,NIST SP 800-53 Rev. 4 CP-2, CP-4, IR-3, IR-4, IR-6, IR-8, IR-9)
Vonatkozó problémák:	SF07
Javaslatok:	JF01
Szabályozás:	3.1.2.1., 3.1.3.8.

Indoklás: Amennyiben a közösségi támogatás elégtelenné válik és szolgáltatást nem lehetséges igénybe venni, a folytonosság biztosításának egyetlen útja a projekt átvállalása (forkolása).

3.2 Védelem (PR)

3.2.1 Azonosítás és hitelesítés, hozzáférés védelem (PR.AC)

Fizikai vagy logikai eszközök csak az arra jogosul személyek, szolgáltatások és eszközök számára hozzáférhetőek. Az eljárások a jogosultságot igénylő műveletek és tranzakciók jogosulatlan elérésének kockázatával arányosak.

VPR01	A FLOSS szoftverek és komponensek frissítéseinek sértetlenségét biztosító kriptográfiai kulcsokat be kell szerezni, biztonságos módon kell tárolni és szükség esetén ellenőrizni.
Keretrendszer:	PR.AC-3 (COBIT 5 APO13.01, DSS01.04, DSS05.03,NIST SP 800-53 Rev. 4 AC-1, AC-17, AC-19, AC-20, SC-15)
Vonatkozó problémák:	ST08, ST09, ST11
Javaslatok:	JK03, JT06
Szabályozás:	3.3.9.5.3.

Indoklás: A FLOSS frissítések sértetlenségét biztosító kulcsok gyakran WoT alapúak PKI helyett. A

frissítések gyakorlatilag távoli elérésnek minősülnek, hiszen lényegében tetszőleges kód futtatható általuk a frissített gépeken. Emiatt rendkívül fontos, hogy a nyílt módszer által favorizált WoT alapú nyilvános kulcsok szükség esetén rendelkezésre álljanak és megbízhatóságuk ellenőrizhető legyen.

3.2.2 Tudatosság és képzés (PR.AT)

A szervezet tagjai és partnerei részesülek kiberbiztonsági képzésben és fel vannak készíve rá, hogy a kiberbiztonsággal kapcsolatos feladataikat a vonatkozó szabályok, eljárások és egyezmények értelmében elvégezzék.

VPR02	A szervezet tudatossági képzése keretében foglalkozik a FLOSS komponensek felhasználásával járó kockázatok és lehetőségek témakörével.
Keretrendszer:	PR.AT-1 (COBIT 5 APO07.03, BAI05.07,NIST SP 800-53 Rev. 4 AT-2, PM-13)
Vonatkozó problémák:	SH04, SH05, SH06, SH09, ST02, ST03
Javaslatok:	JH11, JK07
Szabályozás:	3.1.1.1., 3.1.4.3., 3.1.7.3.

Indoklás: A FLOSS kiberbiztonsági kockázatai eltérhetnek a szokásostól. Különösen fontos tudatosítani, a megfelelő hatékonyságú hibajavítások eléréséhez szükséges módszerek eltéréseit. Az OSSD fejlesztőközösség tagjait más módon kell motiválni, nem lehet számon kérni és kényszeríteni.

VPR03	A szervezet kapcsolatba lép a FLOSS közösséggel és szükség esetén tájékoztatja a felhasználás módjáról és az ezzel járó felelősségről.
Keretrendszer:	PR.AT-3 (COBIT 5 APO07.03, APO07.06, APO10.04, APO10.05,NIST SP 800-53 Rev. 4 PS-7, SA-9, SA-16)
Szabályozás:	3.1.6.6.

Indoklás: A közösségek nem feltétlen vannak tudatában annak, hogy rendszerüket milyen és mennyire kritikus helyeken alkalmazzák. Az információ megosztása segít erősíteni a felelős viselkedést. Különösen fontos lehet kisebb létszámú projektek esetén.

VPR04	Aktívan részt kell vállalni a fejlesztői kommunikációban és hibakeresésben az információáramlás gyorsítása végett, és tisztázni kell a feladatköröket.
Keretrendszer:	PR.AT-3 (COBIT 5 APO07.03, APO07.06, APO10.04, APO10.05,NIST SP 800-53 Rev. 4 PS-7, SA-9, SA-16)
Vonatkozó problémák:	SM01, SM02, SM03, SM07
Javaslatok:	JM03, JM04, JM08
Szabályozás:	3.1.7.1., 3.3.11.6., 3.3.11.8.6.

Indoklás: A közösségnek nincs formális felelőssége. Szerepének fontosságát úgy lehet leginkább tudatosítani, ha a szervezet részt vállal a közösség napi munkájában, folyamatos jelenlétével, esetleg tanácsaival erősítve a közösség tudatosságát. A résztvevők feladatai nem mindig vannak világosan meghatározva, a kommunikáció során törekedni kell ennek tisztázására.

VPR05	A vezetők a szervezeten belül tisztában vannak a FLOSS felhasználás kockázataival, illetve azzal a ténnyel, hogy a FLOSS közvetett felhasználása valószínűleg elkerülhetetlen.
Keretrendszer:	PR.AT-4 (COBIT 5 EDM01.01, APO01.02, APO07.03,NIST SP 800-53 Rev. 4 AT-3, PM-13)
Vonatkozó problémák:	SG01, SH03, SS02
Szabályozás:	3.1.1.4., 3.1.2.2., 3.1.2.3.

Indoklás: A felső vezetés nem feltétlenül van tudatában annak, hogy a szervezet FLOSS termékeket is használ. Az összetett licencelési, integrációs, életciklus menedzsment és inkompatibilitási problémák jelentette kockázatok ismeretében hozható csak meg a megfelelő döntés.

3.2.3 Adatbiztonság (PR.DS)

Az információ és adat kezelése a szervezet kockázatkezelési stratégiájával konzisztens módon történik, odafigyelve a bizalmasság, sértetlenség és rendelkezésre állás kritériumaira.

VPR06	A szabályzatban meghatározott FLOSS komponenseket adott verzión kell rögzíteni (snapshot) és csak a biztonsági frissítéseket kell alkalmazni.
Keretrendszer:	PR.DS-1 (COBIT 5 APO01.06, BAI02.01, BAI06.01, DSS04.07, DSS05.03, DSS06.06,NIST SP 800-53 Rev. 4 MP-8, SC-12, SC-28)
Vonatkozó problémák:	SF10, SF13, SJ03
Javaslatok:	JF05
Szabályozás:	3.1.1.4., 3.1.4.9.3.

Indoklás: A belsőleg felhasznált forráskód adatnak tekinthető. A FLOSS egyedi módszertana gyors egymásutáni kiadásokon alapul, az új kiadásokban végrehajtott módosítások sérülékenységeket vezethetnek be vagy inkompatibilitást okozhatnak. A kockázat egy kiválasztott stabil verziót használatával csökkenthető az esetleges biztonsági hibajavítások folyamatos felügyelete mellett.

VPR07	A forráskód csomagok, bináris formában terjesztett állományok ellenőrző összegeit minden esetben ellenőrizni kell, a szoftvertárolók ilyen irányú képességeit mindig ki kell használni.
Keretrendszer:	PR.DS-2 (COBIT 5 APO01.06, DSS05.02, DSS06.06,NIST SP 800-53 Rev. 4 SC-8, SC-11, SC-12)
Vonatkozó problémák:	ST08, ST09, ST11
Javaslatok:	JT02
Szabályozás:	3.3.10.13.3., 3.3.13.10., 3.3.13.7., 3.3.13.8.

Indoklás: Szinte minden nyílt forrású csomagterjesztési rendszer használ valamilyen integritás-megőrzési megoldást, általában ellenőrző összegeket, illetve a hitelesség biztosítása érdekében digitális aláírásokat. Tekintve, hogy ezek a rugalmasság biztosítása érdekében általában megkerülhetők vagy kikapcsolhatók, a hitelesnek tekintett aláírások köre pedig bővíthető, fontos, hogy a hiteles aláírások köre, azok módosításának lehetősége megfelelően szabályozva legyen. Az ellenőrzési tevékenység elvégzését elő kell írni.

VPR08	A felhasznált DVCS változáskészletének integritását biztosító digitális aláírásokat ellenőrizni kell, illetve ezt biztosító keretrendszert kell használni.

Keretrendszer:	PR.DS-2 (COBIT 5 APO01.06, DSS05.02, DSS06.06,NIST SP 800-53 Rev. 4 SC-8, SC-11, SC-12)
Javaslatok:	JT04
Szabályozás:	3.3.10.13.3., 3.3.13.10., 3.3.13.7., 3.3.13.8.

Indoklás: A modern DVCS rendszerek (pl. git) lehetővé teszik a változáskészletek digitális aláírását, ezáltal azonosítva a változáskészletet jóváhagyó fejlesztőt és biztosítva a változtatás integritását. Ezt a védelmi funkciót csak úgy lehet kihasználni, ha a DVCS felhasználója birtokában van a megfelelő kulcsoknak és aktívan használja a funkciót. (Git esetében például a –verify-signatures opcióval ellenőrizhetők az aláírások merge művelet végrehajtása előtt).

VPR09	A felelős személy vagy automatizált rendszer ellenőrzi, hogy a hibakövető rendszerbe vagy FLOSS szoftvertárolókba visszaolvasztásra kerülő adatok nem tartalmaznak-e érzékeny információkat.
Keretrendszer:	PR.DS-5 (COBIT 5 APO01.06, DSS05.04, DSS05.07, DSS06.02,NIST SP 800-53 Rev. 4 AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4)
Vonatkozó problémák:	SH09, SM05
Javaslatok:	JH10, JK08
Szabályozás:	3.3.10.17., 3.3.10.18., 3.3.11.5.

Indoklás: A helyi módosítások visszavezetése az upstream projektbe számos előnnyel járhat a szervezet számára (lásd: [#sec:Usage]), ugyanakkor kockázati tényezőként figyelembe kell venni a nem kívánt információszivárgás lehetőségét. Megfelelő szabályozás és gyakorlat nélkül könnyen nyilvánossá válhatnak érzékeny adatok (felhasználónevek, hálózati információk, kriptográfiai kulcsok stb). A bizalmasság sérülése gyakran nem is nyilvánvaló, hiszen az érzékeny adat csak egy mellékes forráságon keresztül (pl. feature branch) érhető el. Az információ szivárgás csak akkor akadályozható meg hatékonyan, ha a publikálást végző személy tudatában van a kockázatoknak adott esetben munkáját ellenőrzik is.

VPR10	A közösségi fejlesztésben résztvevő fejlesztők egymás közötti kommunikációja szabályozott és/vagy automatizált módszerekkel szűrt.

Keretrendszer:	PR.DS-5 (COBIT 5 APO01.06, DSS05.04, DSS05.07, DSS06.02,NIST SP 800-53 Rev. 4 AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4)
Vonatkozó problémák:	SK06
Javaslatok:	JK09
Szabályozás:	3.3.11.5.2.

Indoklás: A közösségi fejlesztésben résztvevő fejlesztők elkülönített (közösség által nem látható) kommunikációja nem javasolt, elidegeníti a közösséget és gátolja a közös munkát. A nyílt közösség fejlesztői kommunikációja természeténél fogva publikus. Emiatt fontos szabályozni, hogy milyen információk nem juthatnak el ezekre a csatornákra.

VPR11	A forráskód alapján ellenőrizhető, hogy nem történik-e illegális adatgyűjtés.
Keretrendszer:	PR.DS-5 (COBIT 5 APO01.06, DSS05.04, DSS05.07, DSS06.02,NIST SP 800-53 Rev. 4 AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4)
Javaslatok:	JM07
Szabályozás:	3.1.3.3.3., 3.3.4.3.

Indoklás: Az illegális adatforgalom egy része forgalomanalízis segítségével is beazonosítható, de ha az adott rendszer legális kommunikációt is folytat, az azonosítás rendkívül nehéz lehet. Az információ elrejthető szteganográfiával, időzítésen alapuló információ-átviteli metódusokkal stb. Az egyetlen biztosnak mondható módszer a felhasznált bináris létrehozásához használt forráskód elemzése.

VPR12	A letöltött szabad szoftverek bináris állományainak ellenőrző összegeit minden esetben fel kell használni a sértetlenség biztosítása érdekében.
Keretrendszer:	PR.DS-6 (COBIT 5 APO01.06, BAI06.01, DSS06.02,NIST SP 800-53 Rev. 4 SC-16, SI-7)
Szabályozás:	3.3.6.5.4., 3.3.8.5.2.

Indoklás: A szabad szoftverek könnyű célpontot jelentenek a támadó számára, hiszen egyszerűen módosíthatóak, terjesztésük pedig nincs feltétlen szervezethez kötve és több helyről beszerezhetők. Akkor sem

feltétlen biztosított a sértetlenség, ha a letöltést hiteles forrásból történik, hiszen a szoftvertárolót vagy a kommunikációs csatornát megcélozva a támadó módosíthatta az adatokat. Az állomány sértetlenségének ellenőrzéséhez tehát a hiteles forrásból származó ellenőrzőösszegek helyes használata elengedhetetlen.

VPR13	A szoftvertárházak adatintegritás ellenőrzési képességeit helyesen és folyamatosan használják a komponensek integritás-ellenőrzése során.
Keretrendszer:	PR.DS-6 (COBIT 5 APO01.06, BAI06.01, DSS06.02,NIST SP 800-53 Rev. 4 SC-16, SI-7)
Szabályozás:	3.3.3.6., 3.3.8.5.2.

Indoklás: A bináris állományok mellett trójai kód a forráskódban is könnyen elhelyezhető. Tekintve, hogy a modern fordítási környezetek felépítése meglehetősen komplex és számos külső erőforrást (könyvtárat, segédeszközt), használnak fel, automatikus integritás ellenőrzés nélkül az adatok sértetlenség praktikusan nem biztosítható.

VPR14	A szabályzatban meghatározott szoftvereket csak ellenőrzött forrásból származó, eredeti forráskódból fordítva használ a szervezet.
Keretrendszer:	PR.DS-6 (COBIT 5 APO01.06, BAI06.01, DSS06.02,NIST SP 800-53 Rev. 4 SC-16, SI-7)
Vonatkozó problémák:	ST08, ST10
Javaslatok:	JT05
Szabályozás:	3.1.3.3.3., 3.3.11.8.6.

Indoklás: A bináris és a forráskód logikai azonossága csak úgy biztosítható, ha azt házon belül fordítjuk le vagy megbízható forrásból származik. FLOSS esetén nem mindig garantált a megbízható forrás, ezért szükséges lehet a saját fordítási folyamat alkalmazása. Amennyiben a szoftverforrás megismételhető fordítási folyamatot alkalmaz (reproducible build) ez a lépés elhagyható.

VPR15	Amennyiben a szervezet közvetlenül részt vesz a fejlesztésben, egyértelműen elkülöníti a fejlesztői és stabil változatot. Ha a fejlesztésben közvetlenül nem vesz részt, a beszállítóktól követeli meg ugyanezt.

Keretrendszer:	PR.DS-7 (COBIT 5 BAI03.08, BAI07.04,NIST SP 800-53 Rev. 4 CM-2)
Vonatkozó problémák:	SM08
Szabályozás:	3.3.3.6.

Indoklás: A fejlesztői változat általában sérülékenyebb, új, kevésbé tesztelt funkciókat vezethet be. A szoftver üzemeltetésért felelős személyeknek egyértelműen meg kell tudni különböztetniük a biztonsági tervben előírt stabil változatot a fejlesztői verziótól.

3.2.4 Információ védelemi folyamatok és műveletek (PR.IP)

A szervezet felállította és használja az információs rendszer és eszközeinek védelmét célzó biztonsági rendszabályokat (amelyek a szervezet részeinek célját, hatókörét, felelősségi körét és koordinációját definiálják), folyamatokat és műveleteket.

VPR16	A felhasznált FLOSS komponenseken változtatás csak indokolt esetben és dokumentált módon végezhető. A belső rendszerhez illesztést lehetőség szerint adapter pattern alkalmazásával kell elvégezni.
Keretrendszer:	PR.IP-2 (COBIT 5 APO13.01, BAI03.01, BAI03.02, BAI03.03,NIST SP 800-53 Rev. 4 PL-8, SA-3, SA-4, SA-8, SA-10, SA-11, SA-12, SA-15, SA-17, SI-12, SI-13, SI-14, SI-16, SI-17)
Vonatkozó problémák:	SF11
Javaslatok:	JH06
Szabályozás:	3.1.3.5., 3.3.2.2.

Indoklás: Az eredeti kód módosításával csökken a közösségi támogatás esélye és valószínűleg elveszik az üzleti támogatás lehetősége is. A változtatás új sérülékenységeket vezethet be, és kompatibilitási problémákhoz vezethet. Az adaptern pattern segítségével elkerülhető, hogy az eredeti kódban jelentős módosításokat kelljen végrehajtani.

VPR17	A FLOSS komponensek vagy szoftverek kiválasztásáért felelős személy erre szakosodott partner szolgáltatásait veszi igénybe.
Keretrendszer:	PR.IP-2 (COBIT 5 APO13.01, BAI03.01, BAI03.02, BAI03.03,NIST SP 800-53 Rev. 4 PL-8, SA-3, SA-4, SA-8, SA-10, SA-11, SA-12, SA-15, SA-17, SI-12, SI-13, SI-14, SI-16, SI-17)
Vonatkozó problémák:	SH04, SH05
Javaslatok:	JH04, JJ03, JM05
Szabályozás:	3.3.3.5.

Indoklás: A FLOSS komponensek kiválasztását sokszor egyszerű keresőhasználattal végzik. Ugyanakkor a számtalan változatból és forrásból nagyon nehéz kiválasztani a legmegfelelőbbet. Koránt sem biztos, hogy a legnagyobb letöltésszámmal rendelkező vagy legelsőként felhozott eredmény lesz a megfelelő. A megfelelő tapasztalattal rendelkező partner vagy szolgáltatás segíthet a kiválasztással járó kockázat csökkentésében.

VPR18	A FLOSS kiválasztási folyamatát, körülményeit és eredményét dokumentálni kell.
Keretrendszer:	PR.IP-2 (COBIT 5 APO13.01, BAI03.01, BAI03.02, BAI03.03,NIST SP 800-53 Rev. 4 PL-8, SA-3, SA-4, SA-8, SA-10, SA-11, SA-12, SA-15, SA-17, SI-12, SI-13, SI-14, SI-16, SI-17)
Vonatkozó problémák:	SH05
Javaslatok:	JH05
Szabályozás:	3.3.3.6.

Indoklás: A helyes FLOSS komponens kiválasztása nem triviális feladat. Az ad-hoc választás elkerülése és az ellenőrizhetőség biztosítása végett a kiválasztás folyamatával kapcsolatos információkat meg kell őrizni.

VPR19	A helyben elvégzett fejlesztéseket megfelelő szabályozás mellett a fejlesztők visszavezetik az upstream projektbe.

Keretrendszer:	PR.IP-2 (COBIT 5 APO13.01, BAI03.01, BAI03.02, BAI03.03,NIST SP 800-53 Rev. 4 PL-8, SA-3, SA-4, SA-8, SA-10, SA-11, SA-12, SA-15, SA-17, SI-12, SI-13, SI-14, SI-16, SI-17)
Vonatkozó problémák:	SF12, SH06, SK04
Javaslatok:	JH06
Szabályozás:	3.1.3.3., 3.3.3.8.

Indoklás: Az egyedi változat folyamatos karbantartása időigényes és hibalehetőségeket rejt. Az upstream projekt változásai összeütközésbe kerülhetnek a helyi módosításokkal.

VPR20	A fejlesztést végző részleg licenckezelő célszoftver igénybevételével követi a felhasznált FLOSS komponensek jogi megfelelőségét, külső fejlesztő esetén meg kell követelni a szállított szoftver komponenseinek licenclistáját.
Keretrendszer:	PR.IP-2 (COBIT 5 APO13.01, BAI03.01, BAI03.02, BAI03.03,NIST SP 800-53 Rev. 4 PL-8, SA-3, SA-4, SA-8, SA-10, SA-11, SA-12, SA-15, SA-17, SI-12, SI-13, SI-14, SI-16, SI-17)
Vonatkozó problémák:	SS02, SS03
Javaslatok:	JS04, JS05
Szabályozás:	3.3.3.5., 3.3.3.7.

Indoklás: A rendelkezésre állás biztosítása érdekében a fejlesztett szoftver jogi megfelelőségét folyamatosan ellenőrizni kell. A FLOSS komponensek bonyolult függőségei és a licencelés összetettsége folytán az ellenőrzést megfelelő szoftvertámogatás mellett célszerű végezni. Beszállító igénybevétele esetén a szállított termék megfelelőségének tanúsítása érdekében csatolni szükséges a felhasznált komponensek licenceit.

VPR21	A fejlesztést végző részleg rendelkezzen verziókövetési tervvel.
Keretrendszer:	PR.IP-2 (COBIT 5 APO13.01, BAI03.01, BAI03.02, BAI03.03,NIST SP 800-53 Rev. 4 PL-8, SA-3, SA-4, SA-8, SA-10, SA-11, SA-12, SA-15, SA-17, SI-12, SI-13, SI-14, SI-16, SI-17)
Vonatkozó problémák:	SH06

Javaslatok:	JH07
Szabályozás:	3.3.3.6.

Indoklás: A FLOSS projektek gyors verzióváltásokkal dolgoznak, az új funkciók gyakran csak a fejlesztői verziókban elérhetőek. A fejlesztői verziók stabilitása és biztonsági szintje nem éri el a stabil verziókét, emiatt érdemes lefektetni hogy milyen esetekben milyen verzióváltás végezhető.

VPR22	A szervezet megosztja a FLOSS védelmi intézkedések hatékonyságával kapcsolatos eredményeit más szervezetekkel és részlegekkel.
Keretrendszer:	PR.IP-8 (COBIT 5 BAI08.04, DSS03.04,NIST SP 800-53 Rev. 4 AC-21, CA-7, SI-4)
Szabályozás:	3.1.7.1.

Indoklás: A tudományos szakirodalomban közvetlenül ez az igény nem merül fel, ugyanakkor a tudásmegosztás a nyílt forrás filozófiájával feltétlenül összhangban van, a lehető leghatékonyabb védelmi megoldás kialakítása pedig egyértelműen közös érdek. Tekintve, hogy a dedikáltan FLOSS irányultságú védelmi intézkedések nem terjedtek el széles körben, különösen fontos, hogy az ezekkel kapcsolatos tapasztalatok minél nagyobb publicitást kapjanak.

3.2.5 Karbantartás (PR.MA)

Az ipari vezérlés és információ rendszer karbantartási folyamatai előre definiált szabályok és műveletek alapján kerülnek végrehajtásra.

VPR23	A FLOSS rendszerekkel kapcsolatos hibajegyeket és azokra kapott válaszokat a szervezeten belül központi helyen rögzítik.
Keretrendszer:	PR.MA-1 (COBIT 5 BAI03.10, BAI09.02, BAI09.03, DSS01.05,NIST SP 800-53 Rev. 4 MA-2, MA-3, MA-5, MA-6)
Szabályozás:	3.3.3.6., 3.3.7.2., 3.3.7.2.2.

Indoklás: A FLOSS rendszerek házon belül megvalósított javításait nem minden esetben lehet az eredeti rendszerbe visszavezetni. A közösség által elvégzett javítások különféle hibakövető rendszereken lehetnek nyilvántartva, amelyek elérése nehézkes. Az adott rendszer aktuális biztonsági szintjének értékeléséhez folyamatosan ismerni kell az alkalmazott javításokat és potenciális hibajegyeket, amit egyetlen központi rendszer alkalmazásával lehet biztosítani.

VPR24	A hibajavítások során elvégzett változtatásokat és a verziófrissítések változáslistáit minden módosításánál kereshető módon naplózni szükséges.
Keretrendszer:	PR.MA-2 (COBIT 5 DSS05.04,NIST SP 800-53 Rev. 4 MA-4)
Vonatkozó problémák:	SF10, ST01
Javaslatok:	JT01
Szabályozás:	3.1.3.3., 3.3.3.4.

Indoklás: A felhasznált forrás (harmadik féltől származó) hibajavításait távoli karbantartási eseménynek kell tekinteni. Ennek megfelelően belső rendszerbe átvezetett módosítás idejét, módját, forrását és okát erre alkalmas rendszerben rögzíteni kell. A nyílt forrású rendszerek gyors kiadási gyakorisága miatt gyakran előfordul, hogy egy frissítés (akár fontos biztonsági frissítés) nem kerül átvezetésre az upstream projektbe. A komponensek közötti interdependecia folytán más, nem tervezett komponensfrissítésekre is sor kerülhet ami sérülékenységet vezethet be vagy ütközést okozhat.

3.2.6 Védelmi technológia (PR.PT)

Az alkalmazott technikai védelmi megoldások képesek biztosítani a rendszerek és eszközök biztonságát és ellenállóképességét a vonatkozó rendszabályokban lefektetett elveknek megfelelően.

VPR25	A FLOSS komponenseket fordítás során úgy kell konfigurálni, hogy a lehető legkevesebb szükséges funkciót és függőséget tartalmazzák.
Keretrendszer:	PR.PT-3 (COBIT 5 DSS05.02, DSS05.05, DSS06.06,NIST SP 800-53 Rev. 4 AC-3, CM-7)
Szabályozás:	3.1.3.3.4., 3.3.6.7., 3.3.10.6.

Indoklás: A nyílt forrású szoftverek általában generikusak, széles felhasználási területet céloznak meg, következésképpen sok funkciót és képességet tartalmaznak. Ezek a képességek szinte minden esetben kikapcsolhatók konfigurációs szinten vagy fordítási időben. A minimális szükséges funkciókészlet elvének értelmében valamennyi nem használt képességet ki kell kapcsolni, amely hatékonyabbá teheti a szoftver működését és egyben csökkenti a kikapcsolt funkció és annak upstream projektjeinek esetleges hibáiból származó kockázatot.

VPR26	A forrásból fordított változatokhoz egyedi forkot (branch) kell készíteni, vagy dokumentált módon egyedi fordítási opciókkal kell fordítani olyan módon, hogy az új változat csak a minimálisan szükséges funkciókat tartalmazza.
Keretrendszer:	PR.PT-3 (COBIT 5 DSS05.02, DSS05.05, DSS06.06,NIST SP 800-53 Rev. 4 AC-3, CM-7)
Vonatkozó problémák:	SF07
Javaslatok:	JF04
Szabályozás:	3.3.3.4., 3.3.6.7.

Indoklás: A FLOSS rendszerek gyakran széleskörűen konfigurálhatók, számtalan funkciót tartalmaznak, de ezen funkciók egy része csak fordításkor vagy közvetlenül a forráskódban kapcsolható ki teljesen. A forráskód módosítása csak úgy követhető szisztematikusan, ha annak szoftvertárolója hozzáférhető és abból egyedi forkot készítünk. Egyedi fordítási kapcsolók alkalmazása esetén nincs szükség saját forrásváltozatra, de a későbbi hibás fordítás (regressziós hiba) érdekében a pontos fordítási módszert dokumentálni kell. Néhány operációs rendszer (pl. Gentoo Linux) integrált módon nyújt ilyen szolgáltatást, ennek hiányában alternatív megoldás kialakítása szükséges.

VPR27	Magas biztonsági követelményű feladatokra a szervezet csak formálisan ellenőrizhető forráskódot használ, a kódot a megfelelő automatizált rendszeren ellenőrzi.
Keretrendszer:	PR.PT-3 (COBIT 5 DSS05.02, DSS05.05, DSS06.06,NIST SP 800-53 Rev. 4 AC-3, CM-7)
Vonatkozó problémák:	SS04
Javaslatok:	JM10

Szabályozás:	3.3.3.4., 3.3.3.5., 3.3.6.2.5.

Indoklás: A formális ellenőrzés segítségével biztosítható, hogy a forrásból készített program valóban pontosan a követelmény specifikációban definiált műveleteket végzi el. A nyílt forrás felhasználásának nagy előnye, hogy ha a definiált követelmények illeszkednek a szervezet elvárásaihoz, a program megfelelősége automatizált módszerekkel tanúsítható, nincs szükség tanúsítványokra vagy a tanúsítók iránti bizalomra.

3.3 Felderítés (DE)

3.3.1 Események és eltérések (DE.AE)

A szokásostól eltérő tevékenység nem marad észrevétlen és az események várható hatása felmérésre került.

VDE01	A szervezet csak az engedélyezett szabad szoftver letöltési helyekről és FLOSS szoftvertárolókon keresztül ér el FLOSS binárist vagy forrást.
Keretrendszer:	DE.AE-1 (COBIT 5 DSS03.01,NIST SP 800-53 Rev. 4 AC-4, CA-3, CM-2, SI-4)
Vonatkozó problémák:	SM08, SM09
Szabályozás:	3.1.6.9., 3.3.3.4., 3.3.3.7., 3.3.11.8.6.

Indoklás: A FLOSS rendszerek nyílt szemléletmódja következtében bárki legálisan továbbadhatja, terjesztheti az eredeti szoftvert vagy annak módosított változatát. Rendkívül fontos, hogy a szervezet csak a megbízhatónak tekintett forrásokból szerezzen be alkalmazásokat vagy forráskódot. Az üzleti alkalmazásokkal ellentétben a nyílt forrás továbbterjesztése nem illegális, legitim címekről, hivatalos formában is megoldható, bizonyos esetekben a közösség kifejezetten ilyen célú infrastruktúrát épít ki (ilyen például az Ubuntu PPA rendszere). Elvben a módosított változatokban könnyen elhelyezhető szándékos hátsó kapu vagy trójai szoftver ám ez a támadási forma a kód átláthatósága folytán kockázatos, így valószínűtlen. Ugyanakkor ártatlannak látszó, javításnak álcázott szándékos hiba elhelyezése már sokkal kisebb kockázattal jár. Egy korábbi, ismert sérülékenységgel rendelkező változat használata teljesen ártatlannak tűnhet a

szándékosság nagyon nehezen bizonyítható és ténylegesen véletlen hiba folytán is megtörténhet. A forrást felhasználó szervezet rendszere mindkét esetben kompromittálódik. Az állandó szoftvertárolók és automatizált frissítések használatának különös veszélye, hogy a sérülékenység bármikor – akár az ellenőrzést követően – is bevezethető, később pedig eltávolítható. A kockázat csökkentése érdekében a szervezetnek érdemes ellenőrzése alatt tartania az ilyen jellegű forgalmat.

3.3.2 Folyamatos biztonság felügyelet (DE.CM)

A szervezet folyamatosan figyeli az információs rendszert és eszközeit, és képes beazonosítani az esetleges kiberbiztonsági eseményeket valamint ellenőrzi a védelmi intézkedések hatékonyságát.

VDE02	Hálózatfigyelés segítségével be kell azonosítani a népszerű FLOSS elemek letöltési pontjaira és a forrástárakra irányuló forgalmat.
Keretrendszer:	DE.CM-1 (COBIT 5 DSS01.03, DSS03.05, DSS05.07,NIST SP 800-53 Rev. 4 AC-2, AU-12, CA-7, CM-3, SC-5, SC-7, SI-4)
Szabályozás:	3.3.6.8.3., 3.3.12.5.3.

Indoklás: A szervezet biztonságára káros hatással lehet a fel nem ismert, ellenőrizetlen nyílt forráskód használat. A népszerű letöltési pontok (pl. GitHub, Bitbucket, GitLab stb.) monitorozásával ez a veszély mérsékelhető, illetve felismerhető a nem regisztrált nyílt forrás használat.

VDE03	A letöltött forráskódot a szabályzatban előírt esetekben fordítás vagy futtatás előtt manuális módszerrel ellenőrizni kell.
Keretrendszer:	DE.CM-4 (COBIT 5 DSS05.01,NIST SP 800-53 Rev. 4 SI-3, SI-8)
Vonatkozó problémák:	SM06, SM09
Szabályozás:	3.3.4.3.

Indoklás: A forráskód tartalmazhat hátsó kapukat vagy kártékony kódot. A kód logikai és szintaktikai hibamentessége vizsgálható automatizált módszerekkel, de a szándékosan elhelyezett sérülékenységeket általában csak a kód átnézésével lehet felderíteni amennyiben erre megfelelő erőforrás áll rendelkezésre.

Realisztikus megvalósítására csak rövid parancsfájlok vagy komponensek esetében van esély.

VDE04	A felhasznált FLOSS forráskód módosításait folyamatosan figyelemmel kell kísérni.
Keretrendszer:	DE.CM-6 (COBIT 5 APO07.06, APO10.05,NIST SP 800-53 Rev. 4 CA-7, PS-7, SA-4, SA-9, SI-4)
Szabályozás:	3.3.3.2., 3.3.11.6., 3.3.11.6.2.

Indoklás: A FLOSS automatizált biztonsági frissítéseit külső szolgáltatásnak kell tekinteni, az itt végzett tevékenység ellenőrzést igényel. Tekintettel arra, hogy az automatizált frissítések lényegében ellenőrzés nélkül vezethetnek be sérülékenységeket, a frissítéseket szolgáltató szervezet folyamatos monitorozása szükséges, hogy annak esetleges kompromitálódása és az ellenlépések foganatosítása közötti időablak minimálisra csökkenthető legyen.

3.3.3 Felderítési folyamatok (DE.DP)

A szervezet karbantartja és teszteli felderítési folyamatait, ezáltal felkészülve a szokásostól eltérő eseményekre.

VDE05	A FLOSS komponensek beazonosításáért felelős személy és feladatai világosan definiáltak legyenek.
Keretrendszer:	DE.DP-1 (COBIT 5 APO01.02, DSS05.01, DSS06.03,NIST SP 800-53 Rev. 4 CA-2, CA-7, PM-14)
Vonatkozó problémák:	SG01
Szabályozás:	3.1.1.2., 3.3.3.4.

Indoklás: A FLOSS komponensek egyedi kezelést igényelnek, azonosítatlan használatuk problémákat okozhat. A felhasználók nem feltétlenül vannak tisztában azzal, hogy az általuk alkalmazott komponens egyedi eljárásokat igényelne. A komponensek beazonosításával megbízott személy feladatkörének és jogosultságainak tisztázása segít felbecsülni a fel nem derített FLOSS komponensek használatával járó koc-

kázatot.

VDE06	A FLOSS komponensekben felfedezett sérülékenységet a szervezet az komponensért felelős szervezetnek is jelenti.
Keretrendszer:	DE.DP-4 (COBIT 5 APO08.04, APO12.06, DSS02.05,NIST SP 800-53 Rev. 4 AU-6, CA-2, CA-7, RA-5, SI-4)
Szabályozás:	3.1.1.4.

Indoklás: Azon felül, hogy az eredeti verzió hibamentessége közösségi érdek, ha az eredeti verzió javításra kerül, a szervezetnek nem kell a továbbiakban külön ágon vezetnie a javításait, ami leegyszerűsíti a folyamatot és csökkenti a hibalehetőségeket.

3.4 Válaszlépés (RS)

3.4.1 Analízis (RS.AN)

A hatékony válaszképesség és helyreállító tevékenység biztosítása érdekében a szervezet elemzéseket végez.

VRS06	A szervezet függőségelemzést végez, hogy megállapítsa mely rendszerek érintettek a biztonsági eseményben.
Keretrendszer:	RS.AN-2 (COBIT 5 DSS02.02,NIST SP 800-53 Rev. 4 CP-2, IR-4)
Vonatkozó problémák:	SH08
Szabályozás:	3.1.2.1., 3.1.2.3., 3.3.4.2.

Indoklás: A FLOSS rendszerek összetett függőségi kapcsolatban állnak egymással. A komponensek további alkomponenseket (könyvtárakat, osztályokat, modulokat) használnak. Ezek a függőségek feltérképezhetők, így gyorsan eldönthető, hogy egy adott komponens milyen rendszerekre lehet hatással.

VRS07	A szervezet esemény követő (forensics) elemzése során a forráskódot és a szoftver komponensfüggőségeit is vizsgálja.
Keretrendszer:	RS.AN-3 (COBIT 5 APO12.06, DSS03.02, DSS05.07,NIST SP 800-53 Rev. 4 AU-7, IR-4)
Javaslatok:	JM06
Szabályozás:	3.1.5.8.

Indoklás: A forráskód vizsgálatával eldönthető, hogy a biztonsági eseményt valóban a vizsgált komponens okozta-e. Üzleti megoldások esetében általában nincs lehetőség meggyőződni arról, hogy az adott hibát minden kétséget kizáróan valóban az adott komponens okozta-e. Nyílt forrás esetén az elvi lehetőség adott – bár elvégzése kétség kívül erőforrás-igényes – így a magas biztonsági követelményű területeken érdemes lehet kiterjeszteni a vizsgálatok körét a forráskódra is, hiszen az üzleti rendszerek körére szabott szabályzatok ezt valószínűleg nem tartalmazzák.

VRS08	A felhasznált FLOSS forrásában és hibakövető rendszerében beálló változásokat figyelni kell és/vagy ilyen szolgáltatást nyújtó partner segítségét kell igénybe venni.
Keretrendszer:	RS.AN-5 (COBIT 5 EDM03.02, DSS05.07,NIST SP 800-53 Rev. 4 SI-5, PM-15)
Vonatkozó problémák:	SK01, SM04
Javaslatok:	JM01, JM05
Szabályozás:	3.1.5.8., 3.3.3.2.

Indoklás: A nyílt forrású szoftverek hibajegyei nyilvánosan elérhetőek, következésképpen manuális vagy félautomatikus módszerekkel elemezhetőek. Ez a lehetőség mindenki számára adott, a potenciális támadó viszonylag alacsony erőforrásigénnyel nagy mennyiségű adatot elemezhet új, még nem javított vagy a stabil verzióba még át nem vezetett sérülékenységek és javítások után kutatva. A felhasználó szervezet ugyanilyen módon a hivatalos javítás kiadása előtt tudomást szerezhet a sérülékenységről. A hibajegyek monitorozásának járulékos haszna hogy időben felismerhető a nem megfelelő fejlesztő-bejelentő kommunikáció és lépéseket lehet tenni a probléma elhárítása érdekében. A hibakövető rendszer változásainak követését tehát érdemes beépíteni a sérülékenység bejelentés (fogadás, analízis és válaszlépés) folyamataiba.

3.4.2 Kommunikáció (RS.CO)

A válaszlépéseket a szervezet külső és belső partnereivel együtt koordinált módon hajtja végre.

VRS03	A szervezet együttműködik a közösséggel a hiba elhárítása érdekében. Az együttműködés során betartja a közösség által elvárt technikai és szociális normákat.
Keretrendszer:	RS.CO-4 (COBIT 5 DSS03.04,NIST SP 800-53 Rev. 4 CP-2, IR-4, IR-8)
Javaslatok:	JF01
Szabályozás:	3.1.5.8.

Indoklás: A FLOSS közösséggel való együttműködés eltér a megszokottól. Nincsenek kikényszeríthető határidők, sokat számíthat a kérés pontos megfogalmazása, sőt az udvariasság is. A fejlesztők közvetlen elérhetősége nagy mértékben csökkentheti a válaszidőt, de csak akkor ha az közösség által elvárt norma-rendszer (hibajelentések formátuma, tartalma, szociális normák) betartásra kerül.

VRS04	A hibajegyeket a közösség hibakövető rendszerébe már első alkalommal jó minőségben és reprodukálást lehetővé tevő módon kell felvinni.
Keretrendszer:	RS.CO-4 (COBIT 5 DSS03.04,NIST SP 800-53 Rev. 4 CP-2, IR-4, IR-8)
Vonatkozó problémák:	SK01, SM01, SM02, SM03, SM05, ST03
Javaslatok:	JM13
Szabályozás:	3.3.2.2., 3.3.3.7.

Indoklás: A OSSD közösség jellegzetességei folytán a hibásan vagy rosszul érthető, nem reprodukálható módon felvitt hibajegyek könnyen válasz nélkül maradhatnak. Ellentétben az üzleti támogatással, a hibajegyet kezelő személyt nem kötik szabályok, nem motivált a hibajegy lezárásában, következésképpen a hatékony kommunikáció sokkal nagyobb részben múlik a felhasználón mint üzleti támogatás esetében.

VRS05	A szervezet megosztja fejlesztőközösséggel az általa végrehajtott ellenőrzött válaszintézkedést.
Keretrendszer:	RS.CO-5 (COBIT 5 BAI08.04,NIST SP 800-53 Rev. 4 SI-5, PM-15)

Szabályozás:	3.1.7.1., 3.3.10.17., 3.3.11.6.

Indoklás: A helyes válaszintézkedés publikálása segíti a FLOSS valamennyi felhasználóját. Ugyanakkor a válaszintézkedés nyilvánossá tétele biztonsági kockázatot jelent, mert a nem megfelelően végrehajtott válaszlépés újabb sérülékenységet vezethet be. A várható előnyöket és hátrányokat tehát mérlegelni kell.

3.4.3 Mérséklés (RS.MI)

Az esemény hatásának enyhítése, megoldása és kiterjedésének megakadályozása érdekében a szervezet megfelelő tevékenységeket végez.

VRS09	A forráskód vagy a fordítási környezet megváltoztatásával a szervezet elérhetetlenné teszi a hibát okozó funkciót, így korlátozva a támadás kiterjedését.
Keretrendszer:	RS.MI-1 (COBIT 5 APO12.06,NIST SP 800-53 Rev. 4 IR-4)
Javaslatok:	JS03, JT07
Szabályozás:	3.1.5.8., 3.3.11.7.

Indoklás: A forrás rendelkezésre állása nagyon gyors reakcióidőt tesz lehetővé és a végrehajtás jogi háttere is biztosított. A FLOSS rendszerek általában moduláris szerkezetűek, a hibát okozó funkció gyakran fordítási kapcsolók használatával is kikapcsolható, ami nem igényli a rendszer mélyreható ismeretét. A vészhelyzeti tervnek tartalmaznia kell milyen esetekben és mely rendszereken alkalmazható ez az eljárás.

VRS10	Amennyiben a forrás rendelkezésre áll és a hibát sikerül azonosítani, a szervezet azonnal javítja a hibát vagy kikapcsolja a hibás funkciót.
Keretrendszer:	RS.MI-2 (COBIT 5 APO12.06,NIST SP 800-53 Rev. 4 IR-4)
Javaslatok:	JH02, JS03, JT07
Szabályozás:	3.3.11.6.

Indoklás: A forrás rendelkezésre állása lehetővé teszi az rendszer rövid határidejű módosítását és a mó-

dosítás jogilag is megalapozott. A módosítás a rendszer alapvető szerkezeti ismeretét feltételezi és jelentős szaktudást igényelhet. A vészhelyzeti tervnek tartalmaznia kell milyen esetekben és mely rendszereken alkalmazható ez az eljárás.

3.4.4 Válaszlépés tervezés (RS.RP)

A felderített kiberbiztonsági incidensekre válaszul a szervezet képes végrehajtani a definiált válaszlépéseket és folyamatokat.

VRS01	A biztonsági eseménykezelési eljárásrendben meg kell határozni, hogy milyen események esetén milyen válasz adható a FLOSS komponensek egyedi tulajdonságait figyelembe véve.
Keretrendszer:	RS.RP-1 (COBIT 5 APO12.06, BAI01.10,NIST SP 800-53 Rev. 4 CP-2, CP-10, IR-4, IR-8)
Javaslatok:	JH02
Szabályozás:	3.1.4.2., 3.1.5.8.

Indoklás: FLOSS esetén lehetőség van a forrás azonnali újrafordítására, a támadható képesség kikapcsolására vagy a kód közvetlen módosítására. Nem minden eszköz esetén vállalható azonban az ezzel járó kockázat, továbbá a változtatás folyamatos követése erőforrásigényes. Emiatt dokumentálni szükséges milyen esetekben alkalmazható ez a megoldás.

VRS02	A gyors hibajavítás érdekében meg lehet határozni a hibákat hatékonyan javító fejlesztőket.
Keretrendszer:	RS.RP-1 (COBIT 5 APO12.06, BAI01.10,NIST SP 800-53 Rev. 4 CP-2, CP-10, IR-4, IR-8)
Vonatkozó problémák:	SK02, SM01, SM02, SM03
Javaslatok:	JM02
Szabályozás:	3.1.1.4., 3.1.4.2., 3.1.5.8.

Indoklás: Centralitás vizsgálatok segítségével meghatározhatóak a közösség azon tagjai, akik a legha-

tékonyabban javítják a hibákat illetve ismerik a projektet. A megfelelő személy elérése és motiválása jelentősen csökkentheti a javításhoz szükséges időt. Ugyanakkor az alkotók valós személye nem feltétlenül azonosítható sőt, esetenként azt is nehéz eldönteni, hogy több különféle hozzájárulás egyazon fejlesztőtől származik-e.

3.5 Helyreállítás (RC)

3.5.1 Helyreállítás tervezés (RC.RP)

A szervezet végrehajtja és kezeli a kiberbiztonsági események által érintett rendszerek és eszközök helyreállítását célzó helyreállítási folyamatokat és intézkedéseket.

VRC01	A FLOSS komponens javítását követően a változásokat vissza kell vezetni az upstream tárolóba.
Keretrendszer:	RC.RP-1 (COBIT 5 APO12.06, DSS02.05, DSS03.04,NIST SP 800-53 Rev. 4 CP-10, IR-4, IR-8)
Vonatkozó problémák:	SH06
Javaslatok:	JF01, JF02, JH06
Szabályozás:	3.1.7.1., 3.3.10.17.

Indoklás: A hibajavítás lehető leghamarabb vissza kell kerüljön az upstream projektbe, hogy a későbbi felhasználás ne igényeljen speciális műveleteket. A kód-hozzájárulást a formai és minőségi szabályok betartásával lehetőleg egy ismert közösségi tag vagy saját fejlesztő révén kell bevezetni.

3.6 Részkövetkeztetések

Ebben a fejezetben a NIST Cybersecurity Framework elemeit vetettem össze a korábban meghatározott nyílt forrású sajátosságokkal kapcsolatos javaslatokkal és sérülékenységekkel. Mint kiderült, valamennyi főkategóriában definiálhatók nyílt forrás specifikus elemek, következésképpen a nyílt forrással kapcsolatos

szervezeti szabályozásnak is végig kell követnie a teljes folyamatot.

Az *azonosítás* során meg kell bizonyosodni afelől, hogy a szervezet megfelelő szabályozással rendelkezik-e a FLOSS specifikus kockázatok kezelésére. Fel kell mérni a FLOSS eszközkészleteket, azok forrását és beszerzési útját, meg kell határozni a FLOSS felhasználással kapcsolatos felelősségi köröket, személyeket és a beszállítók szerepét, továbbá ellenőrizni kell, hogy a FLOSS elemek kielégítik-e a kiberbiztonság jogi követelményeit. Az általános kockázatfelmérés és a beszállítói lánc kockázatkezelése során a FLOSS komponenseket és szoftvereket nem javasolt a rendszer többi komponensével azonos módon vizsgálni. A fejlesztési sajátosságokat kezelni képes, dedikált nyílt forrású szoftver elemzési modellekkel kell dolgozni, amely többek között figyelembe veszi a szoftver egyedi életciklusát, a forkolás lehetőségét, a fejlesztében való résztvétel lehetőségét, a javítási és kiadási jellemzőket és egyéb nyílt forrású sajátosságokat.

A *védelem* kialakítása során külön figyelmet kell fordítani arra, hogy a képzés tematikájában a FLOSS sajátosságokból eredő kockázatok is szerepeljenek. A nyílt és zárt fejlesztési modell technológiai eltéréseiből adódóan a szervezetnek a megszokott PKI-n túlmenően készen kell állnia a nyílt modellek által favorizált kriptográfiai eljárások hatékony használatára és a lehetséges beszerzési források nyilvántartására. Ez egyaránt elengedhetetlen a frissítések és a fejlesztői foltok sértetlenségének ellenőrzése során. Amennyiben a szervezet a fejlesztésben való részvétel mellett dönt, szabályozni kell a fejlesztői csatornákon keresztül áramló információt (fejlesztői kommunikáció, forráskód tárolók). A védelmi intézkedések kialakítása során figyelembe kell venni, hogy nyílt forrás esetén lehetőség van a kód azonnali módosítására, a komponensek minimális szükséges funkcionalitással történő fordítására, adott esetben formális ellenőrzésre, amely megnövelheti a rendszer ellenállóképességét csökkentve a kockázatot.

A *felderítés* lépéseinek tervezése figyelembe kell venni, hogy a nyílt modell könnyen elérhető és általában dinamikusan változó terméket eredményez, így szükségessé válhat ezen változások folyamatos monitorozása. A letöltött rövidebb parancsfájlokat érdemes tesztrendszeren vagy manuális módszerekkel ellenőrizni, az automatikus frissítések és forrástárolók elérési pontjai pedig automatizált módszerekkel szabályozhatók és monitorozhatók. Közösségi érdek, hogy a szervezet valamennyi felfedezett sérülékenységet – esetlegesen a javasolt folttal egyetemben – a lehető leghamarabb eljuttassa a fejlesztőközösséghez.

A *válaszlépések* tervezése során meg kell határozni, hogy a nyílt forrás esetében jóval szélesebb válaszlehetőségek közül mely események esetén milyen válasz adható. I. típusú felhasználási gyakorlat felett már

nincs lehetőség a hagyományos támogatás igénybevételére, a FLOSS modell egyedi sajátosságait figyelembe vevő válaszlépésekre van szükség. Ez történhet a hibás funkció kikapcsolásával, belső fejlesztés során megvalósított vagy az eredeti fejlesztők segítségével elvégzett javítás révén. Amennyiben a javítás hagyományos módja nem kielégítő, általában beazonosíthatók és motiválhatók az eredeti kulcsfejlesztők. A hibajegyekkel kapcsolatos válaszlépés tervezése során figyelembe kell venni pszichológiai tényezőket is, FLOSS esetében magasabb szintű, technikai jellegű jelentésekre van szükség.

A *helyreállítás* általános lépései a FLOSS megoldások esetében is alkalmazhatók, de regresszió elkerülése végett különös figyelmet kell fordítani az esetleges javítások forrás projektbe történő visszavezetésére.

Megállapítom, hogy a szervezeti szinten definiálható védelmi intézkedések önmagukban nem képesek a FLOSS felhasználás jelentette veszélyforrások teljes spektrumát lefedni, a ideális védelem eléréséhez állami vagy európai szintű összefogás lenne szükséges.

Összességében kimondható, hogy az I. típusnál magasabb FLOSS felhasználási szint az üzleti termékek esetében bevált gyakorlattól jelentősen eltérő speciális igényeket vet fel, amelyeket a azonosítás, védelem, felderítés, válaszlépések és helyreállítás tervezése során egyaránt figyelembe kell venni.

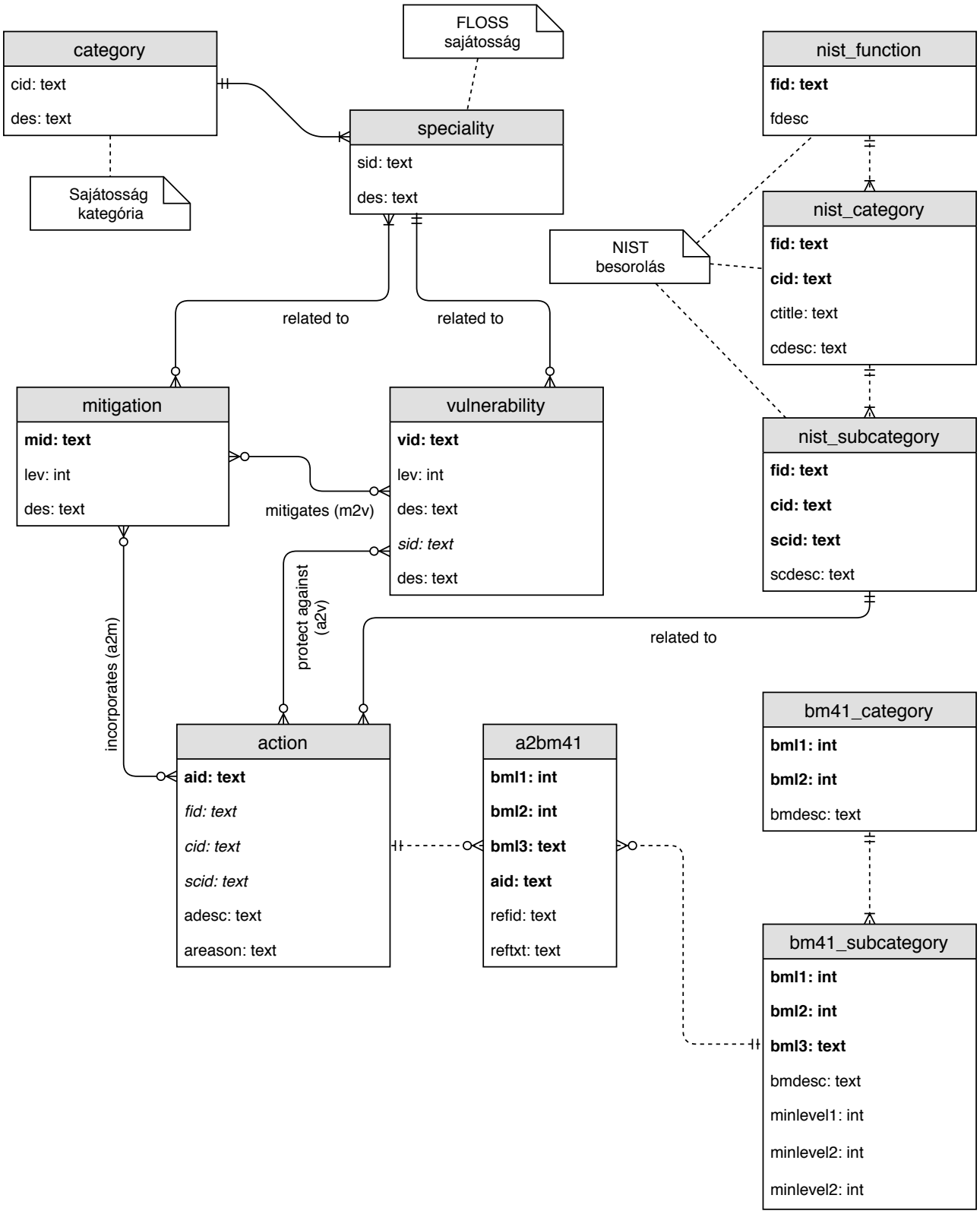
A kifjezetten LIRE vizsgálatához fejlesztett NIST Cybersecurity Framework kategóriái alapján elvégzett analízis során összesen 19 kategóriában tudtam olyan védelmi intézkedéseket definiálni, amelyek képesek növelni az információs rendszer védelmi képességeit. A védelmi képesség pontos mértékének megállapítása meghaladja a kutatás kereteit, valamilyen mértékű hatás léte azonban nem kétséges. A fentiek alapján megállapítom, hogy a H2 hipotézis helytálló.

4 Hazai Szabályozás

A fejezet a hazai szabályozási keretek és a nyílt forráskód felhasználásának kapcsolatrendszerével foglalkozik. Jelenleg hazánkban a Létfontosságú Információs Rendszerelemek információbiztonsággal kapcsolatos előírásait a 41/2015. (VII. 15.) BM rendelet (továbbiakban jogszabály) szabályozza. A harmadik fejezetben definiált kiegészítő védelmi intézkedések egy része egyértelműen megfeleltethető a jogszabályi háttér valamely pontjának, tehát a védelmi intézkedés lényegében az adott követelmény nyílt forrásra vonatkozó realizációjának tekinthető. Ugyanakkor fontos kérdés, hogy a jogszabályban definiált intézkedések a FLOSS rendszerek valamennyi megállapított egyedi sajátosságával kapcsolatos esetleges sérülékenységet lefedik-e, továbbá a hazai szabályozás esetében mely intézkedések során kell FLOSS-al kapcsolatos feladatokra számítani, azaz mit kell tartalmaznia egy esetleges FLOSS specifikus szabályzatnak. A jogszabály egyértelműen fogalmaz: LIRE kritikus rendszerei esetében a legmagasabb (5. szintű) biztonsági osztály irányelvei a mérvadóak, az elemzés során tehát valamennyi intézkedést számításba kell venni.

Ennek a fejezetnek a célja tehát kettős. Egyrészt összegyűjti a szabályozás azon pontjait, amelyek esetén szükség lehet FLOSS specifikus szabályozásra, másrészt a szintézist követően meghatározza azokat a fehér foltokat ahol a szabályozás nem illeszthető megfelelően a FLOSS sajátosságaihoz netán a sajátosságok egyenesen ellentmondásba kerülnek a jogi szabályozással, ellehetetlenítve ezáltal a nyílt forrású rendszer használatát.

A szintézist a korábbi kutatási fázisokban létrehozott relációs adatbázis alapján szisztematikus módszerrel végeztem. Az adatbázis a sérülékenység, összegyűjtött javaslat és javasolt védelmi intézkedés adatait, a jogszabály védelmi szinthez köthető pontjait valamint azok (kutatás során meghatározott kapcsolatait) tartalmazza. Az adatbázis szerkezeti felépítését és definiált kapcsolatait a 4.1 ábra mutatja be. Ez az adatbázis képezheti alapját későbbiekben egy FLOSS szabályozási kérdéseivel foglalkozó szakértői rendszernek.



4.1. ábra: FLOSS analízis-szintézis adatbázisa (szerkesztette a szerző)

Érdemes megemlíteni, hogy a követelmények nem pusztán a Nemzeti Létfontosságú Rendszerelemként azonosított szervezetek számára kötelező érvényűek, az is előírás, hogy ezeknek a felételeknek megfeleljen az információs rendszer "fejlesztője, illetve az információbiztonsági ellenőrzések, tesztelések végrehajtására jogosult szervezet vagy szervezeti egység" is.

Az alábbiakban a jogszabály LIRE-ben felhasznált FLOSS elemek szempontból releváns pontjainak elemzésési eredményeit mutatom be.

4.1 Szervezeti szintű alapfeladatok

A 41/2015. (VII. 15.) BM rendelet előírja, hogy a szervezetnek informatikai biztonsági szabályzattal kell rendelkeznie. Az általam elvégzett elemzés arra enged következtetni, hogy az informatikai biztonsági szabályzatnak mindenképpen javasolt foglalkoznia a FLOSS kérdésével. A szervezetnek egyrészt ismernie kell a saját belső rendszereinek érintettségét, másrészt ismernie és szabályoznia kell a beszállítóitól származó rendszerek nyílt forrásból eredő esetleges problémáit (VID01).

A szabályzatban ki kell térni a FLOSS rendszerek katalogizálásának kérdésére, a FLOSS licenszelés kérdéseire, a beszállítók ellenőrzésének lehetőségeire és az egyedi fejlesztésekben való részvétel szabályozására (VID09). A szervezet így tudja elkerülni az rejtett felhasználást (SG01) és a nemkívánatos adatszivárgásokat (SH08). A FLOSS rendszerek, komponensek minősítése és üzemeltetése egyedi eljárásokat igényel, amelyeket a szabályzatnak szintén rögzítenie kell.

A szabályzatnak akkor is ki kell térnie a nyílt forrás kérdésére, ha a szervezet bizonyíthatóan nem használ ilyen jellegű szoftvert és belső fejlesztést sem végez. A FLOSS komponensek minősítésére nem megfelelő megoldás a COTS rendszerek esetén megszokott módszertan (SH04) és előfordulhat, hogy a nem nyílt forrású rendszerek előállítása során a partnerek által alkalmazott módszertan kritikus hiányosságai miatt a termék biztonsága nem éri el a kívánt szintet illetve a szervezet számára ismeretlen, fekete doboz jellegű tényező marad.

Amennyiben a szervezet a fejlesztésben is részt vállal, a szabályzatnak ki kell térnie az információszivárgás kezelésének eljárásaira és a verzió-ütközések helyes kezelésére is.

Bár a vállalati gyakorlatban a felelősség partnerekre hárítása adott esetben kielégítő megoldást nyújthat, a Létfontosságú Rendszerelemek területén nem elegendő pusztán üzleti szempontokat vizsgálni. A rendszerek kritikus volta és az interdependencia jelensége megköveteli az információs rendszerek biztonságának átlagosnál mélyebb ismeretét.

A jogszabály előírja az információbiztonsággal kapcsolatos szerepkörök és felelősségek kijelölését. Tekintettel a FLOSS rendszerek esetében feltárt jelentős eltérésekre érdemes lehet specifikusan nyílt forrással kapcsolatos szerepkör(öke)t meghatározni. A nyílt forrással kapcsolatos feladatok harmadik félre is átruházhatók, amely esetben ennek végrehajtási és ellenőrzési feltételeit kell a szabályzatban rögzíteni.

A nyílt forrású rendszerek eltérő bánásmódot igényelnek, ami miatt a szervezet képzési szabályzatában is szükséges szerepeltetni a nyílt forrás témáját. Képzésekkel kell tudatosítani a komponens integrációval járó veszélyeket és a biztonságos felhasználáshoz szükséges eljárásokat. A vezetők figyelmét fel kell hívni arra, hogy az OSSD fejlesztőit más módon kell motiválni, a megfelelő kommunikáció szerepe megnövekedhet illetve a FLOSS komponensek minősítésére nem alkalmas a COTS esetén megszokott módszertan.

A jogszabály értelmében valamennyi biztonsági szinten ki kell jelölni az *elektronikus információs rendszerek biztonságáért felelős személyt*. Fontos, hogy a feladatkört betöltő személy ismerje a FLOSS komponensek beazonosításának módszereit és a biztonságos használatukhoz szükséges egyedi eljárásokat. Amennyiben a szervezet több ilyen rendszert is használ vagy nyílt forrású fejlesztésekben is részt vesz, érdemes lehet specifikusan FLOSS személyt kijelölni.

Sajnos a FLOSS esetében a formális szakamberképzés ritka, könnyen előfordulhat, hogy a szervezet nem talál megfelelő szakembert és nincs bevett eljárás a szaktudás tanúsítására, az autodidakta képzés minőségére viszont nincs garancia. A FLOSS felhasználásáért felelős személy feladatkörébe tartozik, hogy meghatározza az érintett rendszerek körét, megállapítsa, hogy a felhasznált FLOSS elemek kielégítik-e a kiberbiztonság vonatkozó jogi követelményeit. Ez adott esetben egyáltalán nem egyszerű feladat, figyelembe véve, hogy a jogrend jelenleg nem tartalmaz FLOSS specifikus elemeket, az ilyen rendszerek ritkán szereznek formális tanúsítást, ami önmagában ellentétet jelent, amennyiben az alkalmazási terület előírja a formális tanúsítás meglétét.

Tekintettel arra, hogy a FLOSS vagy FLOSS elemeket tartalmazó rendszerek egyedi eljárásokat igényelnek, nagyon fontos, hogy az elektronikus információs rendszerek nyilvántartása a FLOSS beazonosítására

is kitérjen. Rejtett FLOSS használat esetén könnyen előfordulhat, hogy nem alkalmazzák a szükséges védelmi eljárásokat (VPR05).

A FLOSS komponensek nyilvántartásának másik oka, hogy a FLOSS komponensekben felfedezett sérülékenységet a komponensért felelős szervezetnek is jelenteni kell amihez ismerni kell annak származási helyét. Az eredeti verzió hibamentessége egyrészt közösségi érdek, másrészt ha az eredeti verzió javításra kerül, a szervezetnek nem kell a továbbiakban külön ágon vezetnie a javításait, ami leegyszerűsíti a folyamatot és csökkenti a hibalehetőségeket (VDE06).

A nyilvántartásnak a komponens listán felül rögzítenie kell a szállító, fejlesztő és karbantartó szervezet vagy személy elérhetőségi adatait. Közvetlen nyílt forrás felhasználás esetén rögzíteni kell a felhasznált verzió azonosítóját és az esetlegesen elvégzett egyedi javításokat (VPR06). Az adott állapot rögzítésével elkerülhető a lehetséges problémák egy része, a frissítési függőségek okozta lavina effektus, ugyanakkor a biztonsági javítások hiánya növeli egy esetleges támadás sikerességét.

Megfelelő nyilvántartás esetén lehetőség van arra, hogy a nyílt rendszerben észlelt kritikus sérülékenység beazonosítását követően haladéktalanul fel lehessen venni a kapcsolatot a javítást potenciálisan leghamarabb elvégző féllel (VRS02).

A nyilvántartásnak érdemes több alternatív elérhetőséget is tárolnia. A nyílt forrás esetében a tényleges alkotók személyazonossága nem mindig határozható meg egyértelműen, az is előfordulhat, hogy a felvett hibajegyek válasz nélkül maradnak, illetve a hiba javításának ellenőrzése is késhet. Amennyiben az elektronikus információs rendszer nyilvántartása a hibabejelentő rendszert, a kapcsolattartó személyt és a javítások ellenőrzéséért felelős személy elérhetőségeit is tárolja a javítás folyamata jelentősen lerövidülhet.

4.2 Kockázatelemzés

Valamennyi biztonsági szinten kötelező elem a kockázatelemzési eljárásrend és a biztonsági osztályba sorolás. Nyílt forráskód alkalmazása esetén a sikeres kockázatelemzés előfeltétele, hogy a kockázatelemzés a FLOSS egyedi sajátosságait figyelembe véve kerüljön végrehajtásra.

Mint korábban arra felhívtam a figyelmet, FLOSS esetében a formális tanúsítás gyakran hiányzik és az üz-

leti termékeknél általában alkalmazott megbízhatósági modellek nem adnak megfelelő eredményt. A felhasznált szoftverek és komponensek kódját és metaadatait tehát olyan biztonság-centrikus megbízhatósági modellekkel kell elemezni amelyek alkalmasak a FLOSS rendszerek minősítésére (VID13). A metaadatok, például a forráskód, hibajegyek vagy a fejlesztői kommunikáció analízisével pontosabb kép alkotható a projekt várható sérülékenységeiről (VID16). A kockázatelemzés során figyelembe kell venni, hogy a nem megfelelő megbízhatósági modell alkalmazása önmagában is kockázati tényező így lehetőség szerint dedikált FLOSS specifikus elemzési módszertant kell alkalmazni (VID18).

Amennyiben a szervezet fejlesztőket delegál a FLOSS közösségbe a kockázatelemzésbe érdemes ezeket a fejlesztőket is bevonni. A megfelelő változat kiválasztása és a változások követése nehéz és időigényes feladat. A FLOSS termékek esetében az alapértelmezett konfiguráció sem feltétlen biztonságcentrikus.

A kockázatelemzés megbízhatóságának mérlegeléséhez ismerni kell a felhasznált dokumentáció és metaadatok minőségét valamint az alkalmazott minősítő eljárás korlátait.

Elvben a kockázatok a forráskód direkt analízisével is felmérhetők (VID14) azonban erre a gyakorlatban csak nagyon egyszerű projektek esetében nyílik valós lehetőség.

A kockázatelemzés során figyelembe kell venni, hogy:

- az eredeti kódbázisba sérülékenységet vezethetnek be, ami az upstream változaton keresztül eléri a szervezet kódbázisát;
- a beszállító vagy közösség monitorozza-e saját komponens projektjeit, az azokban bevezetett sérülékenységek ellen van-e hatékony ellenintézkedés;
- az eredeti kódbázis forkolása esetén a szervezet rendelkezik-e a folyamatos karbantartást biztosító erőforrásokkal;
- a komponensek közötti szoros függőségek miatt a komponens frissítése a részkomponens frissítését is maga után vonja, ami ütközéshez és hibákhoz vezethet;
- a kompatibilitás érdekében helyileg módosított komponens sérülékenységet vezethet be;
- a kis méretű fejlesztőtábor lassíthatja vagy ellehetetlenítheti a hozzájárulásaink időben történő integrálását;
- a fejlesztői vagy tesztelés alatti állapot használata növeli a sérülékenységek esélyét, a kívánt funkció azonban csak egy ilyen verzióban érhető el és a kiadás csúszása miatt a tesztelés alatti változatot

kezdik el használni;

- a szervezet fejlesztőeszközei esetleg nem kompatibilisek a projektben használtakkal, ami lassítja az együttműködést;
- probléma esetén felvett hibajegy válasz nélkül maradhat, javításának ellenőrzése késhet, reprodukálhatatlan ezért figyelmen kívül hagyják;
- a forrástároló, a publikus hibajegyek és a hozzá tartozó foltok folyamatos analízisével a támadó sérülékenységeket fedezhet fel és használhat ki, még azok javítása előtt;
- a fejlesztői kommunikáció monitorozásával a támadó javítás előtt tudomást szerezhet a sérülékenységről;
- egy esetleges támadó a forráskód szisztematikus elemzésével könnyebben fedezhet fel hibákat;
- a hibajegyek hibakeresési csatolmányaként érzékeny információ kerülhet fel a nyilvános hibakereső rendszerre;
- a fejlesztői kommunikáció elemzésével a támadó még javítás előtt tudomást szerezhet a sérülékenységről;
- a nyilvános fejlesztői dokumentáció tervezési hiányosságokat tárhat fel;
- a gyors kiadási sűrűség és munkaigényes frissítés miatt nem vezetik át a biztonsági javításokat a származtatott termékbe;
- a javításra nincs szerződésben biztosított időkorlát, a javítás jelentős mértékben késhet;
- a udvariatlanul kért hibajavítást nem javítják megfelelő sebességgel;
- a biztonság nem feltétlen elsődleges kritérium, amin a szervezet nem tud változtatni;
- az összetett függőségek folytán licencelési problémák merülhetnek fel;
- illetve, hogy kivitelezhető-e a szoftver életpályamodeljének elemzése.

Összességében elemezni kell a nyílt forrás egyedi jellegzetességeinek üzleti hatását és azok valószínűségét. A nyílt forrás nehezen befolyásolható pénzügyi eszközökkel, nem piacvezérelt, továbbá licencelési problémák illetve a forkolás veszélyeztetheti az üzemfolytonosságot (VPR17).

A fentieknek megfelelően a kockázatelemzés során a formális tesztelés kiváltására lehetőleg olyan FLOSS specifikus elemzési módszertant kell alkalmazni, amely kvantitatív eredményeket szolgáltat. Így biztosítható az objektív összehasonlíthatóság. A meghatározott jellemzőket dokumentálni kell és a lehető legszélesebb körben kell felhasználni a rendelkezésre álló metaadatokat (VID19). Tekintettel a forkolás és projekt-

elhagyás kockázatára, a kockázatelemzésnek mindíg ki kell térnie a FLOSS termék életpályamodelljére (VID21) és a javítások reakcióidejére (VID28) is. Késztermék felhasználása esetén javasolt a FLOSS projekt függőségi elemzése, saját fejlesztésbe bevont projekt esetében pedig minden esetben érdemes elvégezni a függőségi elemzést, amely a felhasznált programkönyvtárak által bevezetett esetleges kockázatot is képes figyelembe venni (VRS06).

A kockázatok mérséklésének egyik módja a projekt forkolása, illetve a fejlesztésben történő részvétel. Ezek az esetek azonban újabb kockázati tényezőket vetnek fel, amelyeket az elemzés során szintén figyelembe kell venni.

A jelenlegi jogszabályok nem tartalmaznak FLOSS specifikus kitételeket. Ennek ellenére ellenőrizni kell a jogi megfelelést és a belső szabályzatnak való megfelelést ugyanis a nyílt rendszerek egyedi sajátosságaik miatt nem feltétlenül tudják teljesíteni az előírt követelményeket, például a jogszabály által megnevezett hiteles PKI tanúsítók általi hitelesítést vagy a szabadalomhoz kötött biztonsági megoldásokat.

A kockázatelemzés *végrehajtása* során FLOSS specifikus keretrendszereket kell használni, figyelembe kell venni a közösség életciklusát, a projekt metaadatait és szoftverfüggőségeit. A kockázatelemzést ismételten végre kell hajtani, ha az életciklusban vagy a támogatást nyújtó közösségben jelentős változás áll be. A projekt követésével, a hibajegyek esetleg a forrás elmzésével ezek a változások előrejelezhetők.

4.3 Rendszer és szolgáltatás beszerzés

A szervezet beszerzési eljárásrendjének ki kell terjednie a FLOSS termékek beszállítóira, legyen az akár közvetlenül a közösség avagy üzleti partner. A FLOSS nagyon könnyen hozzáférhető, de ez nem jelenti azt, hogy az ellenőrizetlen felhasználásnak utat kellene engedni. A megbízhatatlan forrásból történő telepítés és szoftverhasználat jelentős kockázatot képvisel.

A beszerzési eljárásrendben ezért ezért célszerű területenként külön, egyértelműen megfogalmazni a FLOSS komponensek és szoftverek elfogadható beszerzési forrásait. Források alatt értendőek a partnerek oldalai, letöltési portálok, szoftvertárolók, a programnyelvek szoftvertárolói és a segítségnyújtó oldalakon fellelhető forráskód részletek is.

Csak olyan forrást szabad megjelölni FLOSS letöltési pontként, amelynek hitelessége kriptográfiai módszerekkel ellenőrizhető. Az eljárásrendben érdemes kitérni az engedélyezett kódfelhasználás részleteire, a keresőmotorral történő FLOSS keresés korlátozásaira (ST10), a bináris változatok felhasználhatóságára és a hitelesnek tekintett szoftvertárolók és csomagkezelő rendszerek listáira.

A szervezetnek meg kell határoznia a FLOSS beszerzéssel kapcsolatos felelősségi köröket és személyeket. A szabályozatlan FLOSS felhasználás mindenképpen kerülendő, következésképpen az ingyenes FLOSS beszerzéseket is szabályozni szükséges. Amennyiben ez elmarad és nincs kijelölt felelős, fenn áll a veszélye a rejtett felhasználásnak.

Az elektronikus információs rendszer fejlesztési környezetére és tervezett üzemeltetési környezetére vonatkozóan érdemes előírni a frissítések és változások ellenőrzésének gyakoriságát, valamint azt, hogy milyen esetekben kell a változtatásokat a közösségi változatba visszavezetni. A visszavezetés legfontosabb előnye, hogy a szervezetnek nem szükséges a továbbiakban külön karbantartania a módosításait, ami által számos inkompatibilitási és regresszós probléma elkerülhető. A visszavezetést akadályozhatja a fejlesztőtábor kis mérete vagy a bizalom hiánya. A kockázat csökkentésének egyik módszere lehet, ha a szervezet támogatást nyújtó partnereket keres komponensek kiválasztása, konfigurálása és üzemeltetése érdekében.

A dokumentációs követelmények megfogalmazásánál érdemes meghatározni, hogy beszállító vagy a fejlesztést végző részleggel szemben milyen FLOSS specifikus dokumentációt vár el a szervezet illetve milyen védelmi intézkedéseket működtet az nyílt forrással kapcsolatos egyedi sérülékenységek csökkentése érdekében. Nem elegendő tehát pusztán a szervezet által használt nyílt forrás felmérése és dokumentálása, meg kell határozni azt is, mely beszállítók építenek FLOSS alapokra vagy használnak FLOSS komponenseket. A jogszabály értelmében a szervezetnek beszerzés során is érvényesítenie kell a védelem szempontjait, ehhez pedig elengedhetetlen, hogy a szervezet ismerje az információs rendszer előállítása során alkalmazott védelmi intézkedéseket. Ennek érdekében érdemes szerződéses követelményként meghatározni a fejlesztő vagy szállító számára, hogy a FLOSS specifikus védelmi intézkedések funkcionális tulajdonságainak a leírását is hozza létre és bocsássa rendelkezésére.

A védelmi intézkedési tervben szerepelnie kell, hogy a megvalósítás során valóban csak ellenőrzött források kerültek felhasználásra és az forráskód ésszerű mértékű ellenőrzése megtörtént. Az elvárt védelmi intézkedési terv része lehet minden további FLOSS specifikus adminisztratív vagy logikai intézkedés amely

a szervezet saját fejlesztéseivel kapcsolatban a jelen fejezetben kifejtésre kerül.

Az elektronikus információs rendszer specifikációjának meghatározását, tervezését, fejlesztését, kivitelezését és módosítását olyan biztonságtervezési elvek mentén kell megvalósítani, amely képes alkalmazkodni a FLOSS fejlesztési metodikájához. Különösképpen jelentős előrelépést jelent – amennyiben a szervezet önálló fejlesztésként használ fel nyílt forrást – ha a szervezet saját fejlesztőit tudja integrálni a fejlesztőközösségbe, így biztosítva a megfelelő lobbierőt, irányítási képességet és információáramlást. Amennyiben a részvétel nem megoldható, az integrációt mindenképpen adapter programozási minta segítségével kell megoldani, máskülönben nem biztosítható a hosszú távú karbantarthatóság vagy csere.

Tekintve, hogy a szervezet külső elektronikus információs rendszereinek szolgáltatásaival kapcsolatban is kötelező megbizonyosodni afelől, hogy azok megfelelnek a szervezet által előírt információbiztonsági követelményeknek, az esetlegesen felmerülő FLOSS specifikus követelményeket a szolgáltatási szerződésben rögzíteni szükséges. A szerződésben vállalt követelmények betartását folyamatosan, külső és belső ellenőrzési eszközökkel ellenőrizni szükséges, illetve független értékelők szolgáltatásait kell igénybe venni. A nyílt forrás kritériumainak esetében ez annyit jelent, hogy a verziófrissítések megtörténtét és a kritikus biztonsági javítások végrehajtását ellenőrizni kell. Ez a megfelelő licensz és verziófájlok ellenőrzésével valamint automatizált hálózati azonosítás segítségével általában megoldható. A folyamatos ellenőrzés biztosítható továbbá a fejlesztőközösségek monitorozásával, saját fejlesztők integrálásával vagy ilyen felelőségi kör meghatározásával, a forráskód illetve a hibajegyek elemzésével és a központi hibajegy adatbázisok monitorozásával.

4.4 Üzletmenet (ügymenet) folytonosság tervezése

Az üzletmenet folytonossági terv két legfontosabb FLOSS specifikus eleme a forkolásra való felkészülés illetve az alternatívák biztosítása.

A nyílt forrás jellemzően nyílt protokollokkal dolgozik, ennek megfelelően az alternatívák használata sok esetben viszonylag kis erőforrásigénnyel megoldható.

A üzletmenet folytonossága veszélybe kerülhet, ha

- a nyílt forrású rendszerelem vagy komponens licencelése megváltozik, ami sérti a szervezet által megkövetelt jogi megfelelőséget avagy lehetetlenné teszi az elvárt szabadalomhoz kötött biztonsági megoldás alkalmazását;
- a közösség felbomlik vagy nem képes tovább biztosítani az elvárt biztonsági követelményeket;
- az upstream változatból kikerül a szervezet számára kritikus szolgáltatás vagy kommunikációs protokoll;

Az ilyen helyzetek kezelése érdekében ki kell jelölni vészhelyzeti szerepköröket, felelősségeket, a kapcsolattartó személyeket. A szervezeten belül történő kijelöléseken túlmenően érdemes lehet a közösségben betöltött szerepeket és felelősségi köröket is nyilvántartani a gördülékenyebb hibaelhárítás érdekében.

Az nyílt forrás alkalmazásának van néhány fontos pozitív vonatkozása is az üzletmenet folytonosság tekintetében:

1. FLOSS esetén lehetőség van a forrás azonnali újrafordítására, a támadható képesség kikapcsolására vagy a kód közvetlen módosítására.
2. A forrás ismeretében egyértelműen beazonosítható a biztonsági problémát (vagy egyéb üzletmenet folytonosságot veszélyeztető) komponens, szolgáltatás vagy funkció.
3. A hiba újrafordítás segítségével saját hatáskörben minimális idő alatt elhárítható, amennyiben a szükséges szakismeret rendelkezésre áll.
4. Közvetlenül meghatározhatók a hibákat hatékonyan és gyorsan javító fejlesztők és a várható javítás ideje. Megfelelő motivációval és direkt kommunikációval a hiba gyorsabban és hatékonyabban orvosolható.
5. Szükség esetén a teljes rendszer forkolható, saját tulajdonba vehető.

Ezzel szemben negatívan hathat az üzletmenet folytonosságra, ha a hibákat hatékonyan javító fejlesztők nem kerültek beazonosításra, nem elérhetők; valamint ha a hibajegy válasz nélkül marad, reprodukálhatatlanság, elégtelen közösségi erőforrások avagy akár udvariatlan megfogalmazás miatt. Megfelelő szakértelem hiányában a szervezet nem tud élni a FLOSS nyújtotta előnyökkel.

A szervezet folyamatos működésre felkészítő képzésének részét kell képezze FLOSS rendszerek és komponensek felhasználásával járó kockázatok és lehetőségek ismertetése (komponens integráció veszélyei, eltérő motiváció, közösségi viselkedés szabályai stb.)

A FLOSS terjesztési módszertana gyors egymás utáni kiadásokon alapul, ugyanakkor az új kiadásokban végrehajtott módosítások sérülékenységeket vezethetnek be vagy inkompatibilitást okozhatnak. A helyreállítási idő minimalizálása érdekében különösen fontos az ismerten működőképes változatok biztonságos tárolása és a helyreállítás tesztelése.

4.5 A biztonsági események kezelése

A szervezet eseménykezelési eljárásának ki kell terjednie az előkészület, észlelést, vizsgálat, elszigetelés és a megszüntetés/helyreállítás lépéseire. Nyílt forrás esetében az előkészületek alatt értendő a korábban már említett forkolásra való felkészülés, a monitorozandó metaadatok meghatározása és elérhetőségek rögzítése. Az észlelés történhet központi adatbázisok alapján, belső monitor rendszer segítségével vagy a közösség saját hibakezelő rendszereinek és egyéb metaadatainak direkt megfigyelésével. A vizsgálat egyedi jellegzetessége a forráskód közvetlen analízise, amire más esetben általában nem nyílna lehetőség. Az elszigetelés és helyreállítás tekintetében a korábban szintén említett gyors módosíthatóságot, újrafordítás lehetőségét érdemes kiemelni.

Az elfogadott és megkívánt eljárásokról a közösséggel való együttműködés szintjéről és módjairól a rendszeresen frissített és ellenőrzött biztonsági esemény-kezelési terv nyújt tájékoztatást.

A FLOSS biztonsági eseményekkel kapcsolatos tényleges előnyeit csak úgy lehet kihasználni, ha a szervezet közvetlenül részt vesz a projekt fejlesztésében. Ezáltal biztosítható a gyors észlelés, az elszigetelés és helyreállítás végrehajtásához szükséges szaktudás.

4.6 Emberi tényezőket figyelembe vevő biztonság

A szervezettel szervezettel szerződéses jogviszonyban álló külső szervezetekkel szemben megállapodásban, szerződésben megkövetelt elvárás, hogy a külső szervezet határozza meg az szervezettel kapcsolatos, az információbiztonságot érintő szerep- és felelősségi köröket, köztük a biztonsági szerepkörökre és felelősségekre vonatkozó elvárásokat is. Közvetlenül nyílt forrás felhasználása esetén ilyesmire nyilvánvaló okokból nincs lehetőség, ugyanakkor a megfelelő tájékoztatást nem szabad elhanyagolni. A szervezetnek

érdemes kapcsolatba lépnie a közösséggel és szükség esetén tájékoztatnia a felhasználás módjáról és az ezzel járó felelősségről. A közösség tagjai nem mindig vannak tudatában, hogy komponensüket vagy rendszerüket kritikus megoldásokban is alkalmazzák, az információ megosztása segít erősíteni a felelősségteljes viselkedést.

A viselkedési szabályok kapcsán elmondható, hogy a szervezetnek a FLOSS kapcsán különös figyelmet kell fordítania a letöltések és ellenőrizetlen nyílt forrás felhasználás okozta veszélyekre, ezért tiltani kell a nem engedélyezett FLOSS szoftvertárolókat és letöltési pontokat. Az ellenőrizetlen – kriptográfiai biztosíték híján ellenőrizhetetlen – publikus letöltési helyek komoly biztonsági kockázatot jelentenek a szervezet információs rendszerére. Nyílt forrás esetében ez fokozottan igaz, hiszen a letöltést végrehajtó személyben az a téves elképzelés alakulhat ki, hogy mivel nyílt forrást használ, azt előtte már több ember ellenőrizte. (Ez egyrészt nem feltétlenül igaz, másrészt a forráskód birtokában rendkívül könnyű a hátsó kapuk beépítése és az ezt tartalmazó változat publikálása.)

4.7 Tudatosság és képzés

A tudatosság és képzés keretében fel kell hívni a figyelmet a nyílt forrás jelentette kockázati tényezőkre, különös tekintettel a népszerű téves elképzelésekre (a nyílt forráskód természetétől fogva biztonságos vagy jobb minőségű) valamint a hitelesség ellenőrzésének megnövekedett fontosságára. A FLOSS kiberbiztonsági kockázatai eltérhetnek a szokásostól. Nyílt forrás esetében nem mindig azonosítható egyetlen, jól definiált forrás szervezet és általában a megszokott PKI infrastruktúra sem elérhető. Alternatív hitelesítési megoldásokra van szükség, PGP, GPG kulcsokkal és hitelesnek minősített kriptográfiai ellenőrzőösszegekkel történik a hitelesség és sértetlenség ellenőrzése.

A jogszabály nem határoz meg külön FLOSS specifikus szervezetrendszert, ugyanakkor fel kell hívni a figyelmet a közösséggel való kapcsolattartás jelentős előnyeire. A fejlesztői kommunikációban való aktív részvétel meggyorsítja a hibakeresést az információáramlást és az esetleges javításokat. Tisztázni kell a feladatköröket is. A közösségnek nincs formális felelőssége, így szerepének fontosságát úgy lehet leginkább tudatosítani, ha a szervezet részt vállal a közösség napi munkájában, folyamatos jelenlétével, esetleg tanácsaival erősítve a közösség tudatosságát.

Tisztában kell lenni azzal is, hogy a nyílt forrású közösségek esetén nincs számon kérhető hibajavítási idő és a javítási idő minimalizálása érdekében be kell tartani a korábban már ismertetett szociális és technikai peremfeltételeket. Az OSSD fejlesztőközösség tagjait más módon kell motiválni, nem lehet számon kérni és kényszeríteni. Az elvégzett javítások visszavezetésének fontosságára és az ezzel kapcsolatos szociális és technikai feltételekre úgyszintén érdemes felhívni a figyelmet.

A FLOSS védelmi intézkedések hatékonyságával kapcsolatos eredmények és tapasztaltok megosztása más szervezetekkel és részlegekkel különösen fontos. A lehető leghatékonyabb védelmi megoldás kialakítása egyértelműen közös érdek, és mivel a dedikáltan FLOSS irányultságú védelmi intézkedések nem terjedtek el széles körben, rendkívül fontos, hogy az ezekkel kapcsolatos tapasztalatok minél könnyebben elérhetők legyenek.

A belső fenyegetések felismerhetősége érdekében a felhasználókban tudatosítani kell a FLOSS megoldások jellegzetességeiből adódó esetleges fenyegetéseket: a könnyű elérhetőség jelentette veszélyeket, a javítások kérésének eltéréseit és az alapértelmezett konfigurálás illetve publikusan elérhető konfigurációs állományok felhasználása által okozott kockázatokat. Fejlesztők esetén a komponens integráció veszélyeit, a folyamatos kiadások okozta kompatibilitási problémák lehetőségét, a közösségi munkájában való részvétel esetén pedig a szociális normák betartásának fontosságát és az információszivárgás lehetőségét.

4.8 Logikai védelmi intézkedések, tervezés

A fizikai védelmi intézkedések jellegüknél fogva nem kapcsolódnak közvetlenül a nyílt forrás fogalmához. A védelmi intézkedések harmadik nagy csoportjával, a logikai védelmi intézkedésekkel kapcsolatban viszont azonosíthatóak FLOSS specifikus elemek, amelyeket a logikai védelmi intézkedések tervezése során figyelembe kell venni.

Tekintettel arra, hogy a nyílt forrás valamilyen szintű használata lényegében elkerülhetetlen, mindenképpen előnyös, ha FLOSS specifikus lépéseket sikerül a biztonságtervezési eljárás folyamataiba integrálni. A rendszerbiztonsági tervben meg kell határozni a nyílt forrással kapcsolatos dokumentációs követelményeket, engedélyezett frissítési methodikát, amelyet meghatározott időközönként felül kell vizsgálni. A cselekvési tervben pedig érdemes rögzíteni a nyílt forrással kapcsolatos egyedi jellegzetességeket, úgymint a

forrásmódosítással kapcsolatos válaszlépések, hibejegykezelés és bejelentés eltérései valamint a jelentési kötelezettségek és kritériumok a közösség felé.

4.9 Rendszer és szolgáltatás beszerzés

Az ebben a szakaszban bemutatott javaslatok csak akkor relevánsak, ha a szervezet saját hatókörben is végez vagy beszerez informatikai szolgáltatást vagy eszközöket illetve ha végez, vagy végeztet rendszerfejlesztési tevékenységet.

A jogszabály előírja, hogy a rendszerfejlesztés során a szervezet vagy a rendszerszolgáltatás fejlesztője folyamatosan nyomon kövesse a elektronikus információs rendszeren (rendszerelemen vagy rendszer szolgáltatáson) elvégzett változásokat. A nyílt forrás egyik prominens jellemzője, hogy a közösségi fejlesztés során valamennyi változtatást valamilyen változáskövető rendszer rögzíti. Amennyiben a szervezet (vagy fejlesztő) ezeket a forrásokat felhasználja, felmerül a kérdés, hogy közösségi tárolás esetén a változások sértetlensége biztosítottnak tekinthető-e ha kizárólag a közösségi forrásra hagyatkozunk. Egyszerű és egyben javasolt megoldás, ha valamennyi felhasznált forrás teljes változási jegyzékét (változáskövető rendszer adatbázisát) a szervezet saját céljaira is archiválja.

Az archiválástól függetlenül a felhasznált FLOSS forráskód módosításait folyamatosan figyelemmel kell kísérni. A FLOSS automatizált biztonsági frissítéseit külső szolgáltatásnak kell tekinteni, az itt végzett tevékenység ellenőrzést igényel. Az ellenőrzést a biztonsági tervben meghatározott biztonsági besorolás szerint kell végrehajtani. Történhet a forrás valamennyi változási deltájának manuális áttekintésével – ami rendkívül idő és szaktudás-igényes folyamat – vagy akár a közösség monitorozásán alapuló.

A fejlesztést végző részleg lehetőség szerint rendelkezzen verziókövetési tervvel. A FLOSS projektek gyors verzióváltásokkal dolgoznak, az új funkciók gyakran csak a fejlesztői verziókban elérhetőek. A fejlesztői verziók stabilitása és biztonsági szintje ugyanakkor nem éri el a stabil verziókét, ezért különösen fontos a felhasznált komponensek verzióinak pontos ismerete.

Fontos, hogy a nyomkövetés a FLOSS eszköz vagy rendszer fejlesztésének teljes életútját végigkísérje egészen a kivonásig.

Mint látható, a közösség, közösségi munka monitorozása lényegében elkerülhetetlen. Ez megvalósítható külső partner szolgáltatása alapján, kifejezetten ilyen célú szerepkör meghatározásával, kulcsfejlesztők bevonásával, illetve – ideális esetben – saját fejlesztők projektbe történő integrálásával.

A változáskövetés tekintetében az alábbiakat kell első sorban szem előtt tartani:

- melyek az engedélyezett szoftvertárolók;
- melyek a megbízhatónak tekintett fejlesztők és nyílt kulcsaik;
- mely tárolókat és milyen időközönként kell archiválni;
- a közösségi szoftvertárolók és a szervezet által használt rendszer kompatibilitása;
- változtatások ellenőrzése;
- változások jóváhagyásának szabályozása;
- a nyílt projekt függőségeinek ellenőrzése.

A szervezeten belüli fejlesztés során általában érdemes saját fejlesztői ágat (branch) létrehozni, amelyet az upstream projekttel rendszeres időközönként ellenőrzött körülmények között szinkronizálni szükséges. Erre technikailag a napjainkban használatos valamennyi jelentősebb változáskövető rendszer alkalmas. Az egyedi fejlesztői ág azt is lehetővé teszi, hogy az új változat csak a minimálisan szükséges funkciókat tartalmazza. A FLOSS rendszerek gyakran széleskörűen konfigurálhatók, de ezen funkciók egy része csak fordításkor vagy közvetlenül a forráskódban kapcsolható ki teljesen. A forráskód módosítása csak úgy követhető szisztematikusan, ha annak szoftvertárolója hozzáférhető és abból egyedi forkot készítünk.

Az saját változat karbantartása során ellenőrizni kell, hogy a szoftvertárházak adatintegritás ellenőrzési képességeit helyesen és folyamatosan használják-e a komponensek integritás-ellenőrzése során. A modern fordítási környezetek felépítése meglehetősen komplex és számos külső erőforrást (könyvtárat, segédeszközt), használnak fel, automatikus integritás ellenőrzés nélkül az adatok sértetlenség praktikusan nem biztosítható.

Ugyanakkor az egyedi “fork” fenntartása rendkívül erőforrásigényes lehet, ezért törekedni kell arra, hogy a publikussá tehető változtatások (javítások és fejlesztések) lehetőség szerint a legrövidebb időn belül visszaolvasztásra kerüljenek az anyaprojektbe.

Következésképpen a szervezet vagy fejlesztő legalább négy különféle ágat kell vezessen párhuzamosan:

az eredeti projekt mester ágát, saját belső fejlesztés módosításait és stabil ágát valamint a visszavezetésre szánt módosítások ágát. Ha a szervezet a fejlesztésben közvetlenül nem vesz részt, a beszállítóktól akkor is meg kell követelni legalább a változások elkülönítését és a felhasznált változat dokumentálását.

Amennyiben a körülmények lehetővé teszik, ideális megoldás, ha a szervezet csak formálisan ellenőrizhető forráskódot használ. A formális ellenőrzés segítségével biztosítható, hogy a forrásból készített program valóban pontosan a követelmény specifikációban definiált műveleteket végzi el, így nincs szükség a fejlesztők vagy tanúsítók iránti bizalomra, sőt tesztekre sem. Sajnos erre a megoldásra – magas idő és szaktudás igénye folytán – csak nagyon ritkán nyílik lehetőség. Természetesen önmagában a formális ellenőrizhetőség nem elegendő, minden változtatás után újra el is kell végezni a formális ellenőrzés lépéseit, hiszen e nélkül a lehetőség minden előnyét elveszti.

A biztonsági tesztelés kapcsán általánosságban elmondható, hogy az egészen kezdetleges projektektől eltekintve valamennyi FLOSS projekt tartalmaz egység-, rendszer-, és regressziós teszteket. Ugyanakkor az integrációs tesztek karbantartása és frissítése általában a fejlesztést végző félre marad. A hiányzó vagy hibás teszteket pótolni kell, ezért a megfelelő projekt kiválasztásának fontos eleme a tesztek minőségének ellenőrzése.

A változáskövetés ki kell terjedjen a felhasznált komponensek és azok részkomponenseinek licensztípusának nyilvántartására is. Nem egyszer előfordul, hogy valamelyik komponens licenctípusa megváltozik és az új licenc nem feltétlenül kompatibilis a korábbival. A nyílt projekt jogi értelemben akár belső ellentmondásokat is tartalmazhat. Hasonló okból a fejlesztést végző partnertől is meg kell követelni a licenctudatosságot, a szállított szoftver esetén a felhasznált komponensek licenclistáját.

A fenti nehézségek miatt előnyös lehet ha a szervezet a FLOSS komponensek beszerzését erre szakosodott szakértők révén szerzi be, és harmadik felet kér fel a komponens minősítésére valamint a biztonsági hiányosságok feltárására. A változáskövetést és licenckövetést erre alkalmas rendszerrel kell végezni.

A fejlesztők jogszabály által előírt oktatási tematikájában érdemes szerepeltetni a nyílt forrással kapcsolatos egyedi jellemzőket és azok hatásait. Nyílt forrással végzett munka során nem feltétlenül elegendő a szervezet által használt belső rendszerek ismerete, a nyílt projektek által alkalmazott népszerű forráskövető rendszerek hiányosságai, eltérései és biztonsági képességeinek ismerete elengedhetetlen, ezért ezeknek az oktatásban is szerepelnie kell. Ugyanez vonatkozik a nyílt licenctípusok ismeretére is.

4.10 Biztonsági elemzés

A szervezetnek előírt időközönként értékelnie kell az elektronikus információs rendszer és működési környezete védelmi intézkedéseit. Nyílt forrás használata esetén az értékelés részben vagy egészben kiterjeszthető a korábban meghatározott függőségekre is, azokat ugyanis a legtöbb projekt további ellenőrzés nélkül használja fel. A FLOSS rendszerek összetett függőségi kapcsolatban állnak egymással. A komponensek további alkomponenseket (könyvtárakat, osztályokat, modulokat) használnak. Ezeknek a függőségeknek a feltérképezésével és rutinszerű ellenőrzésével gyorsan eldönthető, hogy egy adott komponens milyen rendszerekre lehet hatással illetve a szervezet által igényelt funkcionalitás készlettel kapcsolatban állnak-e.

A szervezet által speciális értékelés keretében végrehajtott a biztonságkritikus szoftverelemek forráskód elemzését ki lehet terjeszteni a felhasznált nyílt forráskódú elemekre is. Néhány FLOSS projekt ugyan végez rosszhiszemű felhasználói és belső fenyegetést értékelő teszteket de ez egyáltalán nem általános, következésképpen ezt a szerepet a szervezetnek vagy megbízottjának kell átvállalnia.

Különösen előnyös lehet a nyílt forrás felhasználása olyan biztonságkritikus esetekben ahol az illegális adatgyűjtés elleni védelem kiemelt fontosságú (TCG kompatibilitás). Az illegális kommunikáció szteganográfiával legitim kódba rejthető, időzítés alapú kódolással akár teljesen el is tüntethető. Az egyetlen ténylegesen megbízható megoldás, az eredeti forráskód elemzése.

4.11 Konfigurációkezelés

Az elektronikus információs rendszerekhez a szervezet alapkonfiguráción alapuló, dokumentált és karbantartott konfigurációt kell fenntartania. A nyílt forráskódú elemek gyakran tartalmaznak valamilyen alapkonfigurációt és általában feltételezik, hogy a verzióváltás során a felhasználó az alapkonfigurációt is cseréli, tehát a verzióváltáskor (elsősorban főverzióváltáskor) egyáltalán nem feltétlenül biztosított a visszafele kompatibilitás. A szervezet konfiguráció-kezelő rendszerének erre fel kell készülnie. Az egyszerű karbantartás érdekében a konfigurációt, akárcsak a forrást érdemes változáskövető rendszerben tárolni.

A nyílt forrású csomagok terjesztése esetén általánosan bevett gyakorlat a digitális aláírások használata. A digitális aláírásokat és ellenőrző összegeket minden esetben fel kell használni a sértetlenség biztosítása

érdekében. A szabad szoftverek könnyű célpontot jelentenek a támadó számára, hiszen egyszerűen módosíthatóak, terjesztésük pedig nincs feltétlen szervezethez kötve és több helyről (például tükörszerverek) egyszerűen beszerezhetők. Akkor sem feltétlen biztosított a sértetlenség, ha a letöltést hiteles forrásból történik, hiszen a szoftvertárolót vagy a kommunikációs csatornát megcélozva a támadó módosíthatta az adatokat. Az állomány sértetlenségének ellenőrzéséhez tehát a hiteles forrásból származó ellenőrző összegek helyes használata elengedhetetlen, nem véletlenül a jogszabály elő is írja ezek használatát. Ugyanakkor érdemes kiemelni, hogy a nyílt közösség által használt kriptográfiai eszközök eltérhetnek a megszokottól. Általában nem a PKI infrastruktúrát használnak, hanem a közösségi WoT rendszeren alapulnak. A konfigurációk és csomagok sértetlenségének ellenőrzéséhez tehát a szervezetnek részévé kellene válnia a WoT hálózatának, máskülönben az aláírókulcsok hitelessége nem biztosított.

A konfigurálás során a legszűkebb funkcionalitás alapelvét kell szem előtt tartani (3.3.6.7.). Ennek megfelelően a FLOSS komponenseket fordítás során úgy kell konfigurálni, hogy a lehető legkevesebb szükséges funkciót és függőséget tartalmazzák. A nyílt forrású szoftverek általában generikusak, következésképpen a szükségesnél várhatóan jóval több funkciót és képességet tartalmaznak. Ezek a képességek szinte minden esetben kikapcsolhatók konfigurációs szinten vagy fordítási időben. A minimális szükséges funkciókészlet elvének értelmében valamennyi nem használt képességet ki kell kapcsolni, amely hatékonyabbá teheti a szoftver működését és egyben csökkenti a kikapcsolt funkció és annak upstream projektjeinek esetleges hibáiból származó kockázatot. Ez alapvető eltérés a nem nyílt rendszerek konfigurációjától, tekintve hogy abban az esetben fordítási kapcsolók használatáról egyáltalán nem beszélhetünk azaz a nem kívánt funkcionalitás konfigurációs szinten deaktiválva ugyan de mindenképpen a rendszerben marad.

Az egyedi, csökkentett funkcionalitású változatot a korábban már tárgyalt módon érdemes külön verziókezelői ágon karbantartani.

Az elektronikus információs rendszer védelmének fontos része a jogosulatlan elemek automatikus észlelése. Ennek érdekében az előre meghatározott engedélyezett FLOSS letöltési pontokon kívüli valamennyi egyéb letöltési pontot érdemes monitorozni és szükség esetén megakadályozni a hálózati elérést.

4.12 Karbantartás

A FLOSS rendszerek karbantartása során, hivatalos támogatás híján a karbantartó gyakran közösségi fórumokon keresztül begyűjtött konfigurációs részeket használ fel vagy hasonló úton megszerzett tanácsokat alkalmaz. A visszakereshetőség és nyomonkövethetőség érdekében fontos, hogy a karbantartás változtatásait és lehetőség szerint azok forrását a szervezet valamilyen módon rögzíteni tudja. A FLOSS rendszerekkel kapcsolatos hibajegyeket és azokra kapott válaszokat központi helyen érdemes rögzíteni. Az adott rendszer aktuális biztonsági szintjének értékeléséhez folyamatosan ismerni kell az alkalmazott javításokat és potenciális hibajegyeket, amit egyetlen központi rendszer alkalmazásával lehet biztosítani.

A karbantartás során feltárt biztonságkritikus változtatásokat érdemes az eredeti projektbe visszavezetni – amennyiben ez lehetséges – avagy a közösség figyelmét felhívni a problémára. A FLOSS rendszerek házon belül megvalósított javításait nem minden esetben lehet az eredeti rendszerbe visszavezetni (jogi, technikai vagy szociális okokból), ez esetben külön karbantartói ágat kell vezetni a javításokról.

A karbantartás során automatikus támogatásra kell törekedni. FLOSS esetében a jól definiált kvantitatív értékek határozhatók meg a projekt metaadatai alapján, lehetővé téve a projekt életciklusának automatikus elemzését.

4.13 Adathordozók védelme, azonosítás és hitelesítés

A nyílt forrású szoftverek magas kiadási sűrűsége folytán a statikus adathordozókon történő disztribúció marginális mértékű. Ugyanakkor a helyi szoftvertárház tükröket is adathordozónak lehet tekinteni, amelyre a jogszabály előírja a kriptográfiai védelem biztosítását. A tárházak (szállításnak minősülő) frissítése során úgyszintén előírás az adatok bizalmasságának és sértetlenségének védelme érdekében alkalmazott valamilyen kriptográfiai mechanizmus.

Az interpreter alapú nyelvek esetében általánosan elterjed a komponensfrissítések szoftver csomagkezelő rendszeren keresztül történő elérése (CPAN, hackage, PyPi) amelynek támadásával a felhasznált komponens és így a célrendszer is támadható. Ilyen típusú rendszer használata esetén tehát az ellenőrzésnek a (al)komponensek tárolóinak forgalmára is ki kell terjednie.

Akárcsak a hálózati forgalom és a frissítések során, itt is szembesülünk a nyílt és az üzleti kriptográfiai ellenőrzési rendszerek eltéréseivel. Amennyiben a szervezet közvetlenül a közösségtől szerez be szoftvert vagy komponenseket, kénytelen a közösség által használt WoT alapú kriptográfiai tanúsítványokat használni, máskülönben a kriptográfiai védelemmel kapcsolatos előírások nem teljesíthetőek. A WoT alapú tanúsítványok nem szerezhetők be központosított forrásból, a szervezetnek magának is részévé kell válnia a hálózatnak, vagy megbízhatónak minősített forrásból kell a tanúsítványokat beszereznie.

Minthogy jelenleg ilyen, országos vagy nemzetközi szintű szervezet nem létezik, az esetenkénti részvétel pedig erőforrás igényes, előnyös lenne egy hazai hiteles WoT tárház kialakítása, ahonnan a gyakran használt kriptográfiai tanúsítványok megbízható módon beszerezhetőek átjárást képezve a PKI és a WoT rendszerek között.

A hitelesség szempontjaitól függetlenül a sértetlenség biztosítása érdekében a FLOSS szoftverek és komponensek letöltéséhez és frissítéséhez használt kriptográfiai kulcsokat be kell szerezni, biztonságos módon kell tárolni és szükség esetén ellenőrizni.

A biztonságos tárolás javasolható módja egy helyi kulcs-adatbázis (keyserver) felállítása, amely külső fél számára elérhetetlen, csak az arra jogosult szereplők által módosítható és a szervezet által használt valamennyi tárház kulcsait tartalmazza. A szervezet forrás és szoftvertárházainak konfigurációját úgy kell módosítani, hogy kizárólag a belső kulcs-adatbázisban szereplő kriptográfiai kulcsokat fogadja el hiteles elemként. A WoT alapú kulcsokat PKI struktúrából származó digitális aláírásokkal is ellátva teljesíthető a jogszabályi követelmény.

4.14 Hozzáférés ellenőrzése

A FLOSS szoftver és forrástárházak automatikus és belülről indított frissítéseit külső hozzáférésnek kell tekinteni, ugyanis a frissítési mechanizmuson keresztül az elektronikus információs rendszer működése és beállításai könnyen módosíthatók. Emiatt a hozzáférés ellenőrzésre vonatkozó valamennyi, titkosításra, naplózásra és jogosultságra vonatkozó elvet a frissítések esetében is ugyanúgy kell kezelni, mint bármely más hozzáférés esetén.

Titkosítás tekintetében a már korábban említett módon kell eljárni, központi helyen rögzítve a hitelesség

és sértetlenség ellenőrzéséhez szükséges kulcsokat. Valamennyi frissítési és letöltési hozzáférést naplózni szükséges a forrásként szolgáló tárház címével együtt. Amennyiben a forrástárház (DVCS) digitális aláírásokat alkalmaz a változáskészletek integritásának biztosítására, azokat is ellenőrizni szükséges.

Az információ megosztásának témaköréhez a nyílt forrás két egyedi módon kapcsolódhat: a forrásba visszavezetendő forrásadatbázis elérhetősége és publikálása valamint a hibajegyek rögzítése terén. Mindkét esetben javasolt az információmegosztás automatizált módszerekkel történő segítése, amely alapján a jogosult felhasználók egyszerűen eldönthetik, hogy a megosztásban résztvevő partnerhez rendelt jogosultságok megfelelnek-e az információra vonatkozó hozzáférési korlátozásoknak. Ennek előfeltétele, hogy a forráskódban valamennyi nem publikus információ világosan definiált legyen. Amennyiben a szervezet közvetlenül részt vesz a projekt fejlesztésében, különösen fontos, hogy a publikus és belső fejlesztési ág világosan el legyen választva, közzététel előtt pedig valamennyi publikált információ átnézésre kerüljön. A publikálás lehetőség szerint az erre a feladatra felkészített un. 'Gatekeeper' feladatkörébe kell essen.

A hozzáférések kezelése során a legkisebb elégséges jogosultság elvét kell szem előtt tartani.

4.15 Rendszer és információ sértetlenség

A szervezet által közvetlenül üzemeltetett vagy szolgáltatási szerződés alapján működtetett nyílt forráskódot tartalmazó elektronikus információs rendszer igénybevétele során az alábbiakat kell figyelembe venni.

A kártékony kódok elleni védelem hatáskörébe be kell vonni a frissítési forráscsomagok ellenőrzését is. Nehézséget jelent annak eldöntése, hogy a kiadott bináris csomag valóban az eredeti forrásból készült-e. Ennek automatizált eldöntése jelenleg nem megoldott, de folynak erőfeszítések a megismételhető fordítási folyamatok kialakítása érdekében. Ezek megjelenéséig azonban a hagyományos víruskereső megoldások alkalmazására kell támaszkodni. Alternatív megoldást jelenthet, a biztonsági patch-ek saját hatáskörben történő beépítése, amely jóval megbízhatóbb megoldás, de egyben jelentős szaktudást is igényel.

Az automatizált riasztási rendszer kialakítása során hasznos lehet a nyílt közösség által használt algoritmikusan elemezhető publikus információforrások folyamatos felügyelete. Operációs rendszer esetén a legtöbb disztribúció alkalmaz valamilyen biztonsági riasztási mechanizmust, ezeket az automatizált rendszerbe kell építeni és folyamatosan monitorozni szükséges. Ezen túlmenően a hibakövető rendszer és a

fórumok monitorozásával kulcsszavakra történő vagy mesterséges intelligencia alapú szűréssel idejében észlelhetőek a sérülékenységre utaló jelek.

Ideális esetben a szervezet aktívan szerepet vállal a fejlesztői kommunikációban és hibakeresésben így az információáramlás gyors és megfelelő pontosságúvá válhat.

A válaszlépések kivitelezése során oda kell figyelni rá, hogy a felvitt hibajegy válasz nélkül maradhat vagy figyelmen kívül hagyhatják. Ugyanakkor nyílt forrás közvetlen alkalmazása esetén lehetőség nyílik egy különleges válaszlépés típusra, az újrafordításra. Újrafordítás során a sérülékeny funkciókészlet teljes egészében kiiktatható és a szolgáltatás csökkentett funkcionalitással újraindítható. Ennek automatikus végrehajtása mindamellett nem javasolható, legfeljebb olyan formában, hogy a szervezet szoftverből két alternatív változatot tart fenn, egy erősen csökkentett kritikus funkcionalitás-készlettel rendelkező és egy másik teljes értékű változatot. A teljes értékű változat bővített képességeinek biztonsági problémái esetén automatikusan át lehet térni a csökkentett képességű változatra.

A végrehajtható kódok tekintetében a jogszabály úgy fogalmaz, hogy "Az elektronikus információs rendszer megtiltja az olyan bináris vagy gépi kód használatát, amely nem ellenőrzött forrásból származik, vagy amelynek forráskódjával nem rendelkezik". Nyílt forrás esetében az utóbbi kitétel természetszerűen teljesül, ugyanakkor nem árt szem előtt tartani, hogy forráskód birtoklása önmagában még nem biztosítja, hogy a bináris valóban megfelel annak, sőt, technikailag akár a fordítási környezet is módosíthat a végső bináris állományokon, hátsó kapukat vagy kártékony kódot helyezve el abban. Ennek okán nyílt forrás esetén csak olyan binárisok futtatását szabad engedélyezni, amelyek ismert és ellenőrzött forrásból, ismert és ellenőrzött fordítási környezetben fordítottak le és teszteltek vagy a fordítás folyamata reprodukálható.

4.16 Naplózás és elszámoltathatóság

A korábban már tárgyalt okok miatt érdemes lehet valós idejű riasztást beállítani a nyílt forrású letöltési pontok elérésének figyeléséhez. A riasztások kettős célt szolgálnak. Egyrészt biztosítják, hogy ne valósulhasson meg rejtett, szabályozatlan nyílt forráskód használat, másrészt felderíthetőek a nem ellenőrzött (esetleg nem kívánatos) frissítések (ide értve a felhasznált komponensek frissítéseit is).

A naplózandó események közé kerülhetnek továbbá a nyílt forrású komponens verzióváltások, konfigu-

rációs módosítások, frissítések és a külső forrástárolókkal való kommunikáció (amennyiben a szervezet közösségi fejlesztést is végez).

4.17 Rendszer- és kommunikáció védelem

Az elektronikus információs rendszernek meg kell gátolni a megosztott rendszer-erőforrások útján történő jogosulatlan szándékos vagy véletlen információ áramlást. A nyílt fejlesztési modellben alkalmazott forráskövető és hibakövető rendszer megosztott erőforrásnak tekintendő, ennek megfelelően gondoskodni kell ezen megoldások megfelelő jogosultsági rendszeréről és meg kell határozni az egyes alrendszereken megosztható információk körét.

A fejlesztésben részt vevő munkatársak számára egyedi PGP (GPG) kulcsokat kell létrehozni, amelyet a forráskövető-rendszer változásainak digitális aláírására minden esetben fel kell használni. A kulcsok használata során PKI architektúrával párhuzamosan a GPG rendszer biztonságos használatához szükséges aláírásokat, kulcskiszolgálót és visszavonási listákat is biztosítani kell.

A nyílt forrású szoftverek kiadási ciklusukból adódóan gyakran alkalmaznak frissítéseket. Az adatátvitel sértetlenségének elve értelmében a frissítések ellenőrzését megfelelő erősségű kriptográfiai biztosítékok mellett lehet csak igénybe venni. A nyílt forrású rendszerek minden disztribúciója során ma már általános ezek használata, ugyanakkor ellenőrzésük kihívást jelenthet, részben az aláíró kulcsok hitelességének problematikus ellenőrizhetősége miatt, részben az eljárás nem kötelező volt amiatt. A sértetlenség biztosítása érdekében érdemes a szabályozás kiegészítéseként automatizált módszerekkel kikényszeríteni a frissítések ellenőrző összegeinek és aláírásainak hitelesség ellenőrzését. A modern szoftverfejlesztési gyakorlatban egyre inkább támaszkodnak a komponens alapú fejlesztésekre.

4.18 Részkövetkeztetések

A korábbi kutatási fázisokban meghatározott FLOSS sérülékenységek, javaslatok és az előző fejezetben definiált védelmi intézkedések és a jogszabályi előírások összevetésén alapuló szintézis során a következő megállapításokra jutottam.

- A fizikai védelmi intézkedésektől eltekintve (amelyeket nem vizsgáltam), 18 főkategórián belül 310 (védelmi szinthez kötött) jogszabályi kategóriából összesen 55 esetben volt megállapítható valamilyen FLOSS kapcsolódási pont illetve szükséges védelmi intézkedés. Megállapítom, hogy a FLOSS sajátosságok jogszabályi hatása jelentős. Következésképpen a fejezetben bemutatott szempontokat javasolt figyelembe venni az informatikai biztonsági szabályzat tervezésekor amennyiben a szervezet FLOSS felhasználása vélelmezhető.
- A FLOSS felhasználásnak a LIRE védelmére vonatkozóan egyértelmű előnyei azonosíthatóak.
- A FLOSS sajátosságai miatt összeütközésbe kerülhet a jelen magyar jogi szabályozással, miáltal közvetlen (I. típusnál magasabb szintű) felhasználása ellehetetlenül.

A jelentősebb azonosított FLOSS specifikus problémák az alábbiak:

- A szabályzat elvárja, hogy a szervezet szerződésben rögzítse az információbiztonságot érintő szerep és felelősségi köröket valamennyi külső partner esetén. A nyílt közösségek esetében ez nem kivitelezhető, avagy legalábbis jelentős nehézségekbe ütközik.
- A szerződő féllel szemben támasztott személyi biztonsági követelmények teljesítése problematikus, tekintve, hogy a nyílt közösség természeténél fogva anoním (habár a személyazonosság alternatív módszerekkel általában meghatározható).
- A FLOSS fejlesztő közösség és általában maguk a szoftverek is elsősorban gráfszerkezetű WoT alapú kriptográfiai tanúsítványokat alkalmaznak hierarchikus PKI helyett, amelyet a jelen jogi szabályozás nem kezel. Amennyiben a PKI tanúsítványokat kötelező érvényűnek tekintjük, azzal kizárjuk FLOSS rendszerek jelenetős körét valamint funkcionalitását (pl. automatizált frissítések) amelyek nélkül viszont más követelmények válnak teljesíthetetlenné.
- A biztonsági előírások a fejlesztőre is vonatkoznak, a tranzitívitás folytán végső soron a nyílt közösségnek is teljesítenie kéne valamennyi követelményt, amely néhány triviális területen (pl. forrás megismerhetetlensége, adminisztratív intézkedések) egészen biztosan nem teljesül. Tekintettel a FLOSS ?? fejezetben ismertetett szoftveripari penetrációjának szintjére megállapítom, hogy a témakör további szabályozási figyelmet érdemelne.
- A jogszabály előírja, hogy az információs rendszerre vonatkozó fejlesztői dokumentáció jogosulat-

lanok számára ne legyen megismerhető és módosítható, amely nyílt fejlesztési modellben való aktív részvétel mellett nem teljesíthető.

- A FLOSS projektek nagyon ritkán szereznek jogszabály által is elismert tanúsítást, így nem alkalmazhatók olyan esetekben ahol a tanúsítás megléte előkövetelmény.
- A jogszabály előírja (3.3.3.6.), hogy a fejlesztést végző szervezet biztonsági tervet készítsen és dokumentálja a tervben rögzített lépések elvégzését. A nyílt forrású projektek döntő részénél ez a lépés hiányzik. Az egység-, rendszer- és regressziós tesztelés szintje szintén elmaradhat az elvárttól és nincs lehetőség kikényszeríteni azt.
- A WoT alapú aláírókulcsok használatához a szervezetnek részt kell vennie a WoT hálózatban, a PKI infrastruktúra nem használható hatékonyan. Amennyiben a jogszabály nem teszi lehetővé a WoT használatát (3.3.9.8.2.), a konfigurációkezelés és a forráskód változáskövetés ellenőrzése lényegében lehetetlenné válik.

A fentiek alapján kijelenthető, hogy a nyílt projektek közvetlen felhasználása ütközik a jelen jogszabályban foglaltakkal.

A nyílt közösséggel való együttműködés néhány nyitott kérdést is felvet. Nem ismert, többek között, hogy:

- a közösségi forrástároló elfogadható-e a változtatások hiteles naplójának?
- a PKI architektúra segítségével aláírt digitális aláírókulcsok elfogadhatóak-e hiteles kulcsként (illetve az azt birtokló fejlesztők hiteles forrásként)?
- elfogadható-e szerződéses fejlesztői partnerként (pl. alapítványon keresztül) a nyílt fejlesztői közösség?

A jelenlegi szabályozási környezetben a FLOSS felhasználása nehézkes, a felsőbb biztonsági szinteken a közösséggel együttműködve egyenesen kivitelezhetetlen. Az előnyök kiaknázásához és a védelmi intézkedések egy részéhez aktív közreműködés szükséges, ami gyakran túl nagy feladatot róna a szervezetre.

Ezeket a problémákat elsősorban országos vagy európai összefogással lehet hatékonyan orvosolni. Például, a WoT hálózat szervezetenkénti kialakítása és ellenőrzése feleslegesen erőforrásigényes, ezért szükséges lenne egy országos szintű WoT adatbázisra, amely a gyakrabban használt aláírókulcsok hiteles forrásaként szolgálhatna átmenetet képezve a PKI és a WoT infrastruktúra között. Hasonlóképpen, a forrás megbízha-

tósága (és sértetlensége) biztosítható lenne egy, jogszabály által is hitelesnek tekintett fél által minősített és aláírt verzió központi publikálásával.

A fentiek alapján megállapítom, hogy helyesen definiált védelmi intézkedésekkel a FLOSS sajátosságaiból származó sérülékenysége jelentős része orvosolható, a FLOSS felhasználása előnyös egyedi védelmi intézkedéseket tenne elérhetővé, ugyanakkor a szükséges intézkedések a jelenlegi törvényi szabályozással több ponton ütköznek, következésképpen a H3 hipotézis nem tartható.

5 Összefoglalás

Fontosnak tartom, hogy a FLOSS felhasználó szervezet ne pusztán haszonélvezőként, hanem a közösség felelős tagjaként tevékenykedjen, ami a közösség sőt, végső soron az egész társadalom használt szolgálja. Mint láthattuk a hagyományos fejlesztőközösségek gyenge pontja éppen a formális metódusok és precízen definiált folyamatok hiányában rejlik. A FLOSS felhasználó szervezetek megfelelő célirányos szabályozással éppen ezt a hiányzó láncszemet tudják pótolni, így végső soron a teljes rendszer hatékonysága és biztonsági szintje javul.

5.1 Összegzett következtetések

Értekezésemben a nyílt fejlesztési modellből származó szoftverek (FLOSS) és komponensek felhasználhatósági feltételeit vizsgáltam a Létfontosságú Rendszerelemek információs rendszereiben.

Megállapítottam, hogy a nyílt forráskód felhasználása nem mindig nyilvánvaló, ugyanis a felhasznált komponensek és alrendszerek révén akár többszörösen közvetett módon is megvalósulhat. A többszörösen indirekt felhasználás biztonságra gyakorolt egyértelmű negatív hatása, hogy a sérülékenységek javítása sokkal lassabban megy végbe, a zero-day sérülékenységek akár hosszú ideig is kihasználhatóak maradnak. *Következésképpen, a LIRE informatikai biztonsági szabályzatából akkor sem javasolt kihagyni a nyílt forrás hatásaival foglalkozó intézkedéseket, amennyiben ilyen rendszert a szervezet (látszólag) nem használ.*

Megmutattam, hogy LIRE és a hozzá kötődő LRE közvetlen FLOSS kapcsolódása alapvetően négyféle módon valósulhat meg: viszonteladótól beszerzett üzleti termékként vagy annak részeként, közvetlenül a közösségtől bináris vagy módosítás nélküli futtatható formában, belső elírások szerinti fordítás által, tehát

módosított/ellenőrzött forráskód formában, végül a fejlesztői közösség részeként, a fejlesztésben való aktív részvétellel.

Az kapcsolat és együttműködés felsőbb szintjein a zárt forrás esetében ismeretlen védelmi lehetőségek nyílnak meg, de ezzel párhuzamosan a lehetséges sérülékenységek száma is növekszik.

A nyílt forrás sajátosságainak rendszerszemléletű elemzése során megállapítottam, hogy a FLOSS és fejlesztési metodikája bizonyos területeken olyan egyedi jellemzőkkel rendelkezik amelyek pozitív és negatív irányban is befolyásolhatják az informatikai biztonságot. Feltártam, hogy a felmerülő problémák egy részére a kutatóközösségnek már van valamilyen javasolt megoldása, de számos kérdés továbbra is nyitott és gyakran szabályozatlan marad. A FLOSS szabályozása jelenleg nem kiforrott.

Megállapítom, hogy a nyílt forrású fejlesztési modell alapján termékek és komponensek olyan egyedi sajátosságokkal bírnak, amelyek befolyásolhatják a felhasználó szervezet informatikai biztonsági szintjét. Ennek alapján a H1 hipotézist elfogadtam.

A kifejezetten LRE elemzésekre szabott NIST Cybersecurity Framework módszertanát alapul véve meghatároztam azokat a védelmi intézkedéseket, amelyekkel a FLOSS specifikus sérülékenységek jelentette kockázat mérsékelhető. Valamennyi fő kategóriában azonosíthatóak voltak nyílt forrás specifikus elemek, következésképpen *a nyílt forrással kapcsolatos szervezeti szabályzatnak végig kell követnie a teljes folyamatot*.

Megállapítottam, hogy a COTS (dobozos) felhasználásnál magasabb szintű FLOSS felhasználási szint esetében *az üzleti termékek esetében bevált gyakorlattól jelentősen eltérő speciális lehetőségek és igények merülnek fel, amelyeket az azonosítás, védelem, felderítés, válaszlépések és helyreállítás tervezése és megvalósítása során egyaránt figyelembe kell venni*.

Az ellenintézkedések egy része a nyílt fejlesztési modell sajátosságaiból adódó sérülékenységek kockázatának mérséklését célozza, néhány esetben azonban olyan új típusú ellenintézkedéseket is sikerült azonosítani, amelyek alkalmazása zárt forrású termékek esetében elképzelhetetlen.

A 3. fejezetben definiált ellenintézkedések alapján a H2 hipotézist elfogadtam.

A Létfontosságú Rendszerelemek hazai biztonsági követelményeinek és a nyílt forrású modell egyedi sajátosságainak összevetése során feltártam, hogy a klasszikus FLOSS termékek nem felelnek meg maradék-

talanul a legmagasabb biztonsági szint által definiált elvárásoknak, ezért a jelen szabályozás alapján csak harmadik féltől származó termékekbe építve használhatóak. *A H3 hipotézist tehát el kell vetni.*

5.2 Eredmények érvényessége

A kutatás tervezése során igyekeztem körültekintően eljárni, hogy az elért eredmények érvényessége a kitűzött céloknak megfelelő szintű legyen. Úgy gondolom a rendelkezésemre álló erőforrásokhoz képest sikerült átfogó és teljes körű képet adni a feldolgozott témáról, de az általam alkalmazott módszertannak és az egyszemélyben végzett kutatómunkának természetszerűen vannak bizonyos korlátai. Az itt elért tudományos eredmények csak ennek kontextusában értelmezhetők. Az alábbiakban ezeket a korlátokat ismertetem.

5.2.1 Belső érvényesség

Az adatgyűjtést és az analízist egyedül végeztem, aminek következtében óhatatlanul bizonyos mértékű szubjektivitás terheli az eredményeket. Ennek tudatában munkám során igyekeztem a szubjektivitás mértékét minimálisra csökkenteni. A vonatkozó kutatásmódszertani elveknek megfelelően a kutatás minden szakaszát előre lefektetett objektív szabályrendszer használata mellett szisztematikusan végeztem, egyértelmű osztályozási struktúrát alkalmaztam, az egyes szakaszokat eltérő időpontban felülvizsgáltam és iterációs módszerrel folyamatosan javítottam.

Ugyanakkor a kutatás egyes részeiben kénytelen voltam bizonyos mértékig saját műszaki tapasztalataimra és józan ítélőképességre is hagyatkozni, ami természetszerűen szubjektív. Ilyen kutatási szakasz a FLOSS sajátosságok osztályozási rendszere vagy a vonatkozó sérülékenységek és kontrollok végső meghatározása és osztályozása.

Az eredményeket befolyásolhatta az adatgyűjtés során kiválasztott publikációk halmaza. A kiválasztási halmaz okozta torzítások kivédése érdekében igyekeztem több forrásból nagy mennyiségű releváns információt feldolgozni. A második fejezetben bemutatott keresőszöveget úgy választottam meg, hogy a lehető legnagyobb de még feldolgozható mennyiségű kiindulási adattal dolgozhassak. A publikációk kiválasztása és osztályozása manuálisan történt, ami újabb szubjektív elemet visz a kutatásba. Praktikus lett volna

egy második kutató segítségével validálni a kiválasztási halmaz helyességét de erre nem volt lehetőségem. Mindamellett a beazonosított biztonsági sajátosságok forrásainak átfedései azt tanúsítják, hogy az eredmények kellően megalapozottak.

Véleményem szerint a megállapított sajátosságok alapján levont következtetések érvényessége a kutatás célkitűzéseinek megfelelő, ellenben a sajátosságok teljeskörűségére nincs biztosíték. Sajnos maga a kutatási módszer sem tette ezt lehetővé, hiszen elképzelhető, hogy a tudományos közösség sem derítette fel a FLOSS sajátosságok egy részét, sőt már a felderített jellemzők arányának becslése is problematikus.

Összefoglalva, az itt bemutatott eredmények nem teljes körűek de a FLOSS kutatás jelenlegi állapotához mérten megalapozottak és jól használhatók a meglévő külső és belső FLOSS szabályozások kiegészítésére és pontosítására.

5.2.2 Külső érvényesség

A kutatómunka céljai közt szerepelt, hogy az elért eredmények a megcélzott területen jól általánosíthatóak legyenek. Ennek biztosítására rendszerszemléletű megközelítést alkalmaztam, a lehető legnagyobb látókörrel megközelítve a problémát folyamatosan szűkítve azt a célterület felé. Az feldolgozott a információ sajátosságai és módszertan miatt azonban a generalizálás során körültekintően kell eljárni.

- A forrásként szolgáló publikációk többsége elsősorban a jelentősebb nyílt forrású közösségekkel és azok termékeivel foglalkozik, így a megállapítások egy része csak azokra érvényes bizonyítható módon. A nyílt modell közösségeinek jó része kicsi, gyakran egy személyes projekt, amelyek vizsgálatával csak kevesebb tanulmány foglalkozik. Az elemzések során több ilyen publikációt is feldolgoztam és igyekeztem figyelembe venni az ezekben azonosított különbségeket, de a nagy projektekre koncentráló forrásanyag túlsúlya biztosan torzítja az eredményt.
- A FLOSS sajátosságok sokkal inkább általános gráfszerű kapcsolatban állnak egymással semmint az itt bemutatott hierarchiában. Más kutató valószínűleg némiképpen más osztályozási rendszert alkalmazott volna. Ennélfogva a FLOSS taxonómia mint eredmény nem tekinthető abszolút osztályozási rendszernek, pusztán egy változat a számtalan lehetséges közül. A kutatás végső célját tekintve ennek a korlátnak véleményem szerint nincs jelentős hatása, hiszen a sajátosságok osztályba

sorolását a szisztematikus vizsgálat biztosítása végett vezettem be azért, hogy az egyes kategóriákon módszeresen végig lehessen menni és a duplikációkat egybe lehessen olvasztani. Bármely más csoportosítás, más úton ugyan, de hasonló végső eredményre vezetett volna. Minthogy az osztályozás nagy mennyiségű anyagot dolgoz fel, a lehetőségekhez mérten teljes körű, így más kutatások számára is használható keretrendszert biztosít.

- A Létfontosságú Rendszerelemek szigorú belső szabályozása folytán nemigen volt lehetőségem betekinteni az ottani folyamatok részleteibe. Személyes találkozók keretében sikerült ugyan bizonyos mennyiségű információt összegyűjtenem (OKF, OMSZ) de a kritikus szabályozási folyamatok egészére nem volt rálátásom. A kapott információk publikálására sem kaptam engedélyt, így a kutatás elsősorban a nagyvállalatok szabályozási mintáin és az USA szövetségi információs rendszerek számára készült publikus kiadványain alapul. Véleményem szerint ezek a lehetőségekhez képest jól lefedik a megcélzott területet, de az eredményeket ennek fényében kell értelmezni.
- A kutatás a beazonosított problémákhoz nem határoz meg valószínűségeket, következésképpen kockázatkezelési célokra csak úgy használható, ha ezeket saját vagy külső forrásból sikerül meghatározni.

6 Új tudományos eredmények

- Szisztematikus és rendszerszemléletű feldolgozási módszer segítségével meghatároztam és egységes rendszerbe foglaltam a nyílt forrású fejlesztési modellből származó szoftverek és komponensek sajátosságait.
- Az új osztályozási módszer mentén elvégzett analízis módszerével igazoltam, hogy a nyílt forrású fejlesztési modell és az ilyen módszerrel létrehozott komponensek olyan egyedi sajátosságokkal bírnak, amelyek befolyásolhatják a felhasználó szervezet informatikai biztonsági szintjét.
- Az egyedi FLOSS sajátosságok és a Létfontosságú Rendszerelemek minősítésére alkalmas NIST Cybersecurity Framework összevetésén alapuló deduktív stratégia alapján olyan lehetséges ellenintézkedéseket definiáltam, amelyek mérsékelni képesek a FLOSS felhasználó szervezet által viselt kockázatokat.
- A Létfontosságú Renszerelemek biztonsági követelményeit szabályozó hazai jogszabályi környezet (41/2015 BM rendelet), a nyílt forrású modell egyedi sajátosságainak és a definiált védelmi intézkedések összevetésével feltártam, hogy a klasszikus FLOSS termékek nem felelnek meg maradéktalanul a legmagasabb biztonsági szint által definiált elvárásoknak, ezért a jelen szabályozás alapján csak harmadik féltől származó termékekbe építve használhatóak.

7 Ajánlások

Az értekezésemben összefoglalt kutatási eredmények felhasználhatók minden olyan Létfontosságú Információs Rendszerelem védelmi intézkedéseinek tervezéséhez ahol nyílt forrású rendszereket használnak, használni fognak vagy azokkal valamilyen módon – például beszállítókon keresztül vagy SLA egyezmény keretében – kapcsolatba kerülnek. Véleményem szerint ez hosszabb távon elkerülhetetlen, ezért az itt bemutatott eredmények információs és képzési szinten hasznos segítséget nyújthatnak a szervezet biztonságért felelős szakemberei számára, valamint segíthetnek kiegészíteni a szervezet információ-biztonsági szabályzatait.

Nist Security Framework, a ISO 27000 sorozat és a magyar jogrend (41/2015. (VII. 15.) BM rendelet) egyaránt előírja a rendszeres biztonsági tudatossági képzést. Ezt a képzést csak szakképzett személy végezheti. Ajánlom eredményeimet a szakképzést nyújtó szervezetek és szakemberek figyelmébe, illetve a szakemberek képzését végző szervezetek figyelmébe kiegészítő anyagként.

Az elkövetkező kutatások célja lehet az egyes FLOSS specifikus problémák súlyának meghatározása és támadási valószínűségekhez rendelése. A javasolt védelmi intézkedések jelen formájukban csak ad-hoc módon alkalmazhatók, hiszen azok szükségességét a szabályozást megelőző kockázatbecslés során kellene megállapítani, amit támadási valószínűségek és súlyossági értékek hiányában nemigen lehet precízen kivitelezni. Természetesen a kockázatkezelés nehezen általánosítható, így egy olyan módszertanra lenne leginkább szükség, amellyel ezek az értékek az azonosított problémák esetében becsülhetők.

8 A témakörben készült publikációim

Lektorált, magyar nyelvű szakmai folyóiratcikkek

1. Mészáros Gergely: FLOSS fejlesztői közösségek hatása az informatikai biztonságra, In: Koncz, István; Szova, Ilona (szerk.) A tudomány szolgálatában című IX. Ph.D. - Konferencia előadásai (Budapest, 2014. október. 29.): Elektronikus könyv, Budapest, Magyarország : Professzorok az Európai Magyarországért Egyesület, (2014) pp. 28-36. , 9 p.
2. Mészáros, Gergely: Nyílt forráskódú rendszerek biztonsági kérdései, BOLYAI SZEMLE XXII : 1 pp. 63-76. , 14 p. (2013)
3. Mészáros Gergely: Elosztott verziókezelés a közigazgatásban, Hadmérnök IX : 3 pp. 191-206. , 16 p. (2014)
4. Mészáros Gergely: Információs rendszerek fenyegetéseinek képesség alapú osztályozása, Társadalom és Honvédelem XVII : 3-4. pp. 215-227. , 13 p. (2013)
5. Mészáros, Gergely: Szun-Ce elvei a digitális világban, Hadmérnök VIII. : 2 pp. 377-388. , 12 p. (2013)
6. Mészáros, Gergely ; Hufnagel, Levente: Filtering outliers in OMTK Data Set, ANNUAL NEWS OF THE SZENT ISTVÁN UNIVERSITY YBL MIKLÓS FACULTACY OF BUILDING SCIENCES 7 : 1 pp. 42-46. , 5 p. (2007)
7. Mészáros, Gergely: Magyarországi szántóföldi tartamkísérletek adatainak mesterséges intelligencia alapú elemzési lehetőségei Tudományos Közlemények Szent István Egyetem Ybl Miklós Műszaki Főiskolai Kar 1 pp.32-34 (2004)

Lektorált, angol nyelvű szakmai folyóiratcikkek

1. Mészáros, Gergely: Auditing Community Software Development, YBL Journal of Built Environment 3 : 1-2 pp. 26-33. , 8 p. (2015)
2. Mészáros, Gergely: Yield Prediction Based on Neural Networks, Annual News of The Szent István Unniversity YBL Miklós Faculty of Building Sciences 3: pp. 90-95., 6 p. (2005) 5.Mészáros, Gergely: Human-Computer Interaction through Hand Gestures, Annual News of the Szent István University Ybl Miklós Faculty of Building Sciences pp.38-41., 4 p. (2003)

Idegen nyelvű konferencia kiadványban megjelent cikkek

1. Mészáros, Gergely: Lessons of Transparent Collaboration: Comparison of E-Government and Software Developer Communities, In: Balthasar, Alexander; Golob, Blaž; Hansen, Hendrik; Müller-Török, Robert; Nemeslaki, András; Pichler, Johannes; Prosser, Alexander (szerk.) Central and Eastern European e|Dem and e|Gov Days 2016 : Multi-Level (e)Governance : is ICT a means to enhance transparency and democracy? Vienna, Ausztria : Austrian Computer Society, (2016) pp. 383-392. , 9 p.
2. Mészáros, Gergely: Security impacts of community based software development pp. 325-336. In: Alexander, Balthasar; Blaž, Golob; Hendrik, Hansen; Balázs, Kőnig; Robert, Müller-Török; Alexander, Prosser (szerk.) CEE e|Dem and e|Gov Days 2015 : Time for a European Internet? Wien, Ausztria : Austrian Computer Society, (2015) p. 629
3. Mészáros, Gergely: Pattern Classification via Neural Networks, In: Bergmeister, K (szerk.) Proceedings of the 3rd International PhD Symposium in Civil Engineering : Vol.2 Wien, Ausztria : Fleck Druck Gmbh, (2000) pp. 475-478. , 4 p.

Lektorált, magyar nyelvű előadás

1. Mészáros, Gergely: Katasztrófavédelem és nyílt forrás, In: Kiss, Dávid; Orbók, Ákos (szerk.) A haza szolgálatában 2014 konferencia rezümékötet, Budapest, Magyarország : Nemzeti Közszolgálati Egyetem, (2014) pp. 66-68. , 3 p.
2. Mészáros, Gergely: Nyílt forráskód létjogosultsága a kormányzati rendszerekben, In: Keresztes, Gábor (szerk.) Tavaszi Szél, 2013 : Spring wind, 2013. 1-2. kötet, Budapest, Magyarország : Doktoranduszok Országos Szövetsége, (2013) pp. 46-54. , 9 p.
3. Mészáros, Gergely: Kritikus infrastruktúrákban felhasznált nyílt forráskódú rendszerek auditálási kérdései, In: Szakál, Béla (szerk.) Intézeti tudományos konferencia, Budapest, Magyarország : Avernim, (2012) pp. 91-97. , 7 p.
4. Mészáros Gergely: GIS rendszertervezés nyílt forráskódú alapokon, In: Márkus, Béla (szerk.) GIS-open 2011 : Megfelelni az új kihívásoknak Székesfehérvár, Magyarország : Nyugat-magyarországi Egyetem Geoinformatikai Kar, (2011) pp. 55-64. , 10 p.

Lektorált, idegen nyelvű előadás

1. Mészáros, Gergely: IoT security and education, In: Talata, István (szerk.) Matematikát, Fizikát és Informatikát Oktatók 41. Országos Konferenciája : MAFIOK 2017, Budapest, Magyarország : Szent István Egyetem Ybl Miklós Építéstudományi Kar, (2017) pp. 1-7., 7 p.

Magyar nyelvű kivonat

1. Mészáros Gergely: Funkcionális programozás geometriai alkalmazásai, In: Attila, Bölcskei; Gyula, Nagy (szerk.) GeoGra : 2012. 01. 20-21.: SZIE Ybl Miklós Építéstudományi Kar, Budapest, Magyarország : Szent István Egyetem Ybl Miklós Építéstudományi Kar Ábrázolás és Számítástecnikai Tanszéke, (2012) p. 26 , 1 p.

Hivatkozott irodalom

[1] Bocsok Viktor, Borbély Zsuzsa: Kritikus Infrastruktúra Üzemeltetés a Jövőben - Törvénytől a Megoldásig. In: *Florian express*, 2013. Vol. 22, no. 3, pp. 78–85, ISSN 1215-492.

[2] E. Erturk: A Case Study in Open Source Software Security and Privacy: Android Adware. In: *Internet Security (WorldCIS), 2012 World Congress On*, IEEE, 2012. pp. 189–191. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6280226>

[3] GlobalStats: Mobile Operating System Market Share Worldwide. In: *StatCounter Global Stats*. [2018-09-7] Retrieved: <http://gs.statcounter.com/os-market-share/mobile/worldwide>

[4] GlobalStats: Mobile Operating System Market Share Asia. In: *StatCounter Global Stats*. [2018-09-7] Retrieved: <http://gs.statcounter.com/os-market-share/mobile/asia>

[5] B. Weinberg: The Internet of Things and Open Source (Extended Abstract). In: *Interoperability and Open-Source Solutions for the Internet of Things*, Springer, Cham, 2015., Lecture notes in computer science. pp. 1–5. ISBN 978-3-319-16545-5 978-3-319-16546-2.

[6] GlobalInfoResearch: Hadoop Market Size, Share And Forecast To 2022. In: *MarketWatch*. 2017. [2018-08-24] Retrieved: <https://www.marketwatch.com/press-release/hadoop-market-size-share-and-forecast-to-2022-2018-06-27>

[7] D. Spinellis, V. Giannikas: Organizational adoption of open source software. In: *Journal of Systems and Software*, 2012 március. Vol. 85, no. 3, pp. 666–682, ISSN 01641212. DOI 10.1016/j.jss.2011.09.037.

[8] Browser Market Share.. [2018-08-24] Retrieved: <https://netmarketshare.com/browser-market-share.aspx>

[9] R. Hewapathirana, P. Amarakoon, J. Braa: Open Source Software Ecosystems in Health Sector: A Case Study from Sri Lanka. In: *Information and Communication Technologies for Development*, Springer, Cham, 2017 május 22. pp. 71–80.

[10] E. Petrinja, A. Sillitti, G. Succi: Adoption of Oss Development Practices by the Software Industry: A Survey. In: *Open Source Systems: Grounding Research*, Springer, 2011., pp. 233–243. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-24418-6_16>

[11] N. Munga, T. Fogwill, Q. Williams: The Adoption of Open Source Software in Business Models: A Red Hat and IBM Case Study. In: *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, New York, NY, USA: ACM, 2009. pp. 112–121.

[12] C. Bouras, A. Filopoulos, V. Kokkinos, S. Michalopoulos, D. Papadopoulos, G. Tseliou: Policy recommendations for public administrators on free and open source software usage. In: *Telematics and Informatics*, 2014 május. Vol. 31, no. 2, pp. 237–252, ISSN 07365853. DOI 10.1016/j.tele.2013.06.003.

[13] Haig Zsolt: *Információ – Társadalom – Biztonság*, Budapest: NKE Szolgáltató Kft., 2015. ISBN 978-615-5527-08-1.

[14] *2013. évi L. törvény*. 2013 április 25. Magyar Közlöny Lap- és Könyvkiadó. Retrieved: <http://www.kozlonyok.hu/nkonline/MKPDF/hiteles/MK13069.pdf>

[15] C. Krasznay: *A Magyar Elektronikus Közigazgatási Szolgáltatások Komplex Információvédelmi Megoldásai*. Zrínyi Miklós Nemzetvédelmi Egyetem Hadtudományi kar Katonai Műszaki Doktori Iskola, 2011.

[16] K.F. Punch: *Developing Effective Research Proposals*, S.l.: Sage, 2007. ISBN 978-1-4129-2126-8.

[17] S. Leshem, V. Trafford: Overlooking the conceptual framework. In: *Innovations in Education and Teaching International*, 2007 február. Vol. 44, no. 1, pp. 93–105, ISSN 1470-3297, 1470-3300. DOI 10.1080/14703290601081407.

[18] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson: Systematic Mapping Studies in Software Engineering. In: *12th International Conference on Evaluation and Assessment in Software Engineering*, sn, 2008. pp. 68–77. [2015-07-14] Retrieved: <http://www.rbsv.eu/courses/rmtw/mtrl/SM.pdf>

[19] K. Petersen, S. Vakkalanka, L. Kuzniarz: Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update. In: *Information and Software Technology*, 2015 augusztus 1. Vol. 64, pp. 1–18, ISSN 0950-5849. DOI 10.1016/j.infsof.2015.03.007.

[20] B. Kitchenham: Procedures for Performing Systematic Reviews. In: *Keele, UK, Keele University*, 2004. Vol. 33, no. 2004, pp. 1–26, [2015-02-25] Retrieved: <http://tests-zingarelli.googlecode.com/svn-history/r336/trunk/2-Artigos-Projeto/Revisao-Sistematica/Kitchenham-Systematic-Review-2004.pdf>

[21] Mészáros Gergely: Security impacts of community based software development. In: *CEE e|Dem and e|Gov Days 2015*, 2015: 2015., ISBN 978-2-85403-308-0.

[22] Joint Task Force Transformation Initiative: NIST SP 800-53r4: *Security and Privacy Controls for Federal Information Systems and Organizations*. National Institute of Standards and Technology, 2013. [2018-08-20] Retrieved: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf>

[23] *2012. Évi CLXVI. Törvény A Létfontosságú Rendszerek És Létesítmények Azonosításáról, Kijelöléséről És Védelméről*. 2012.

[24] Vass Sándor: Az Elektronikai Berendezéseinket Fenyegetõ Terrortámadások És Az Ellenük Való Védekezés Kérdései. In: *Nemzetvédelmi Egyetemi Közlemények*, 2006. Vol. X. évf., no. 3., pp. 228–236,

[25] T. Bonnyai: *A Kritikus Infrastruktúra Védelem Fogalmi Rendszere, Hazai És Nemzetközi Szabályozása*. 2011. Katasztrófavédelmi Tudományos Tanács. [2013-01-6] Retrieved: <http://www.vedelem.hu/letoltes/tanulmany/tan382.pdf>

[26] B. Olga, S.C. Petronela, D. Radu, M. Constantin: CONSIDERATIONS REGARDING THE COMPANIES'OBLIGATIONS TOWARDS THE DIRECTIVE 2008/114/CE CONCERNING CRITICAL INFRASTRUCTURES. In: *Fascicle of Management and Technological Engineering*, 2010. Vol. IX, ISSN 1583-0691.

[27] Bonnyai Tünde: *A Kritikus Infrastruktúra Védelem Elemzése a Lakosságfelkészítés Tükrében*. 2014.

[28] Richard Stallman: Why Open Source misses the point of Free Software.. [2018-08-12] Retrieved: <https://www.gnu.org/philosophy/open-source-misses-the-point.html>

[29] A. Capiluppi, K.-J. Stol, C. Boldyreff: Exploring the Role of Commercial Stakeholders in Open Source

Software Evolution. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 178–200. [2015-10-21] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-33442-9_12>

[30] F. Weikert, D. Riehle: A Model of Commercial Open Source Software Product Features. In: *Software Business. From Physical Products to Software Services and Solutions*, 2013., Lecture Notes in Business Information Processing, 150. pp. 90–101. ISBN 978-3-642-39336-5.

[31] W. Scacchi, T.A. Alspaugh: Designing Secure Systems Based on Open Architectures with Open Source and Closed Source Components. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 144–159. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-33442-9_10>

[32] M. Umarji, S.E. Sim, C. Lopes: Archetypal Internet-Scale Source Code Searching. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 257–263. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_21>

[33] M. Squire, A.K. Smith: The Diffusion of Pastebin Tools to Enhance Communication in FLOSS Mailing Lists. In: *Open Source Systems: Adoption and Impact*, Springer, Cham, 2015 május 16. pp. 45–57.

[34] B. Napoleão, K. Felizardo, É. Souza, N. Vijaykumar: Practical similarities and differences between Systematic Literature Reviews and Systematic Mappings: A tertiary study. *The 29th International Conference on Software Engineering and Knowledge Engineering*, 2017 július 5. pp. 85–90. [2018-09-14] Retrieved: <http://ksiresearchorg.ipage.com/seke/seke17paper/seke17paper_69.pdf>

[35] Klaas-Jan Stol: *Supporting Product Development with Software from the Bazaar*. Doktori (PhD) értekezés. UNIVERSITY OF LIMERICK, 2011. Retrieved: <http://staff.lero.ie/stol/files/2011/12/stol_phd_2011.pdf>

[36] A. Al-Ajlan: The Evolution of Open Source Software Using Eclipse Metrics., IEEE, 2009 június. pp. 211–218.

[37] ALI BABAR, Muhammad, DINGSØYR, Torgeir, LAGO, Patricia és VAN VLIET, Hans (szerk.): *Software Architecture Knowledge Management*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02373-6 978-3-642-02374-3.

[38] B. Xu, D.R. Jones, B. Shao: Volunteers' involvement in online community based software development. In: *Information & Management*, 2009 április. Vol. 46, no. 3, pp. 151–158, ISSN 03787206.

DOI 10.1016/j.im.2008.12.005.

[39] W.H.M. Theunissen, A. Boake, D.G. Kourie: In Search of the Sweet Spot: Agile Open Collaborative Corporate Software Development. In: *Proceedings of the 2005 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, Republic of South Africa: South African Institute for Computer Scientists and Information Technologists, 2005. pp. 268–277.

[40] Z.S. Wubishet: Understanding the Nature and Production Model of Hybrid Free and Open Source Systems: The Case of Varnish. In: *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference On*, IEEE, 2009. pp. 1–11. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4755628>

[41] C. Melian, M. Mähring: Lost and Gained in Translation: Adoption of Open Source Software Development at Hewlett-Packard. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 93–104. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_8>

[42] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, D.M. German: Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub. In: *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, Piscataway, NJ, USA: IEEE Press, 2015. pp. 574–585.

[43] E. Capra, C. Francalanci, F. Merlo, C. Rossi-Lamastra: Firms' involvement in Open Source projects: A trade-off between software structural quality and popularity. In: *Journal of Systems and Software*, 2011 január. Vol. 84, no. 1, pp. 144–161, ISSN 01641212. DOI 10.1016/j.jss.2010.09.004.

[44] Ø. Hauge, S. Ziemer: Providing Commercial Open Source Software: Lessons Learned. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 70–82. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_8>

[45] K.Y. Sharif, M. English, N. Ali, C. Exton, J. Collins, J. Buckley: An empirically-based characterization and quantification of information seeking through mailing lists during Open Source developers' software evolution. In: *Information and Software Technology*, 2015 január. Vol. 57, pp. 77–94, ISSN 09505849. DOI 10.1016/j.infsof.2014.09.003.

[46] D. Spinellis: A Tale of Four Kernels. In: *Proceedings of the 30th International Conference on Soft-*

ware Engineering, ACM, 2008. pp. 381–390. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1368140>

[47] C. Izurieta, J. Bieman: The Evolution of FreeBSD and Linux. In: *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ACM, 2006. pp. 204–211. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1159765>

[48] M. Soto, M. Ciolkowski: The QualOSS Process Evaluation: Initial Experiences with Assessing Open Source Processes. In: *Software Process Improvement*, Springer, 2009., pp. 105–116. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-04133-4_9>

[49] M. Theunissen, D. Kourie, A. Boake: Corporate-, Agile-and Open Source Software Development: A Witch's Brew or An Elixir of Life? In: *Balancing Agility and Formalism in Software Engineering*, Springer, 2008., pp. 84–95. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-85279-7_7>

[50] J.A. Bergstra, P. Klint: About „trivial" software patents: The IsNot case. In: *Science of Computer Programming*, 2007 február. Vol. 64, no. 3, pp. 264–285, ISSN 01676423. DOI 10.1016/j.scico.2006.09.003.

[51] S.S. Bahamdain: Open Source Software (OSS) Quality Assurance: A Survey Paper. In: *Procedia Computer Science*, 2015. Vol. 56, pp. 459–464, ISSN 18770509. DOI 10.1016/j.procs.2015.07.236.

[52] A. Capiluppi, T. Knowles: Software Engineering in Practice: Design and Architectures of Floss Systems. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 34–46. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_5>

[53] J.W.C. Llanos, S.T.A. Castillo: Differences between Traditional and Open Source Development Activities. In: *Product-Focused Software Process Improvement*, Springer, 2012., pp. 131–144. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-31063-8_11>

[54] M. Aberdour: Achieving Quality in Open-Source Software. In: *Software, IEEE*, 2007. Vol. 24, no. 1, pp. 58–64, [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4052554>

[55] E. Damiani, C.A. Ardagna, N. El Ioini: OSS Security Certification. In: *Open Source Systems Security*

Certification, Springer, 2009., pp. 1–36. [2015-11-27] Retrieved: <http://link.springer.com/content/pdf/10.1007/978-0-387-77324-7_5.pdf>

[56] A. Schofield, G.S. Cooper: Levels of Formality in FOSS Communities. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 337–342. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_38>

[57] J. Wang, P.C. Shih, Y. Wu, J.M. Carroll: Comparative case studies of open source software peer review practices. In: *Information and Software Technology*, 2015 november. Vol. 67, pp. 1–12, ISSN 09505849. DOI 10.1016/j.infsof.2015.06.002.

[58] N. Bettenburg, A.E. Hassan, B. Adams, D.M. German: Management of community contributions: A case study on the Android and Linux software ecosystems. In: *Empirical Software Engineering*, 2015 február. Vol. 20, no. 1, pp. 252–289, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-013-9284-6.

[59] S.A. Ajila, D. Wu: Empirical study of the effects of open source adoption on software development economics. In: *Journal of Systems and Software*, 2007 szeptember. Vol. 80, no. 9, pp. 1517–1529, ISSN 01641212. DOI 10.1016/j.jss.2007.01.011.

[60] V.K. Gurbani, A. Garvert, J.D. Herbsleb: A Case Study of a Corporate Open Source Development Model. In: *Proceedings of the 28th International Conference on Software Engineering*, New York, NY, USA: ACM, 2006. pp. 472–481.

[61] I. Stamelos: Management and Coordination of Free/Open Source Projects. in: RUHE, Günther és WOHLIN, Claes (szerk.), In: *Software Project Management in a Changing World*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014., pp. 321–341. ISBN 978-3-642-55034-8 978-3-642-55035-5.

[62] P.M. Bach, J.M. Carroll: FLOSS UX Design: An Analysis of User Experience Design in Firefox and OpenOffice. Org. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 237–250. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_21>

[63] B. Johansson: Diffusion of Open Source ERP Systems Development: How Users Are Involved. In: *Governance and Sustainability in Information Systems. Managing the Transfer and Diffusion of IT*, Springer, 2011., pp. 188–203. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-24148-2_12>

[64] A.J. Ko, P.K. Chilana: Design, Discussion, and Dissent in Open Bug Reports. In: *Proceedings of the 2011 iConference*, ACM, 2011. pp. 106–113. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1940776>

[65] K. Ven, H. Mannaert: Challenges and strategies in the use of Open Source Software by Independent Software Vendors. In: *Information and Software Technology*, 2008 augusztus. Vol. 50, no. 9-10, pp. 991–1002, ISSN 09505849. DOI 10.1016/j.infsof.2007.09.001.

[66] P. Laurent, J. Cleland-Huang: Lessons Learned from Open Source Projects for Facilitating Online Requirements Processes. In: *Requirements Engineering: Foundation for Software Quality*, Springer, 2009., pp. 240–255. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02050-6_21>

[67] J. Noll: Requirements Acquisition in Open Source Development: Firefox 2.0. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 69–79. [2015-11-27] Retrieved: <http://link.springer.com/content/pdf/10.1007/978-0-387-09684-1_6.pdf>

[68] N. Iivari: Discursive construction of „user innovations" in the open source software development context. In: *Information and Organization*, 2010 április. Vol. 20, no. 2, pp. 111–132, ISSN 14717727. DOI 10.1016/j.infoandorg.2010.03.002.

[69] A. Raza, L.F. Capretz, F. Ahmed: An open source usability maturity model (OS-UMM). In: *Computers in Human Behavior*, 2012 július. Vol. 28, no. 4, pp. 1109–1121, ISSN 07475632. DOI 10.1016/j.chb.2012.01.018.

[70] J. Li, R. Conradi, O.P.N. Slyngstad, C. Bunse, M. Torchiano, M. Morisio: An Empirical Study on Decision Making in Off-the-Shelf Component-Based Development. In: *Proceedings of the 28th International Conference on Software Engineering*, ACM, 2006. pp. 897–900. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1134446>

[71] C.S. Wright: Software, Vendors and Reputation: An Analysis of the Dilemma in Creating Secure Software. In: *Trusted Systems*, Springer, 2011., pp. 346–360. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-25283-9_23>

[72] A. Khanjani, R. Sulaiman: The Aspects of Choosing Open Source versus Closed Source. In: *Computers & Informatics (ISCI), 2011 IEEE Symposium On*, IEEE, 2011. pp. 646–649. [2015-11-24] Retrieved:

<http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958992>

[73] C. Subramaniam, R. Sen, M.L. Nelson: Determinants of open source software project success: A longitudinal study. In: *Decision Support Systems*, 2009 január. Vol. 46, no. 2, pp. 576–585, ISSN 01679236. DOI 10.1016/j.dss.2008.10.005.

[74] J. Vuorinen: Ethical codes in the digital world: Comparisons of the proprietary, the open/free and the cracker system. In: *Ethics and Information Technology*, 2007 február 28. Vol. 9, no. 1, pp. 27–38, ISSN 1388-1957, 1572-8439. DOI 10.1007/s10676-006-9130-2.

[75] R. Kemp: Current developments in Open Source Software. In: *Computer Law & Security Review*, 2009 november. Vol. 25, no. 6, pp. 569–582, ISSN 02673649. DOI 10.1016/j.clsr.2009.09.009.

[76] S.S. Levine, M.J. Prietula: Where and When Can Open Source Thrive? Towards a Theory of Robust Performance. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 156–176. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_13>

[77] D. Lorenzi, C. Rossi: Assessing Innovation in the Software Sector: Proprietary vs. FOSS Production Mode. Preliminary Evidence from the Italian Case. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 325–331. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_29>

[78] J. Ellis, J.-P. Van Belle: Open Source Software Adoption by South African MSEs: Barriers and Enablers. In: *Proceedings of the 2009 Annual Conference of the Southern African Computer Lecturers' Association*, ACM, 2009. pp. 41–49. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1562746>

[79] J. Noll, W.-M. Liu: Requirements Elicitation in Open Source Software Development: A Case Study. In: *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, New York, NY, USA: ACM, 2010. pp. 35–40.

[80] R. Zilouchian Moghaddam, M. Twidale, K. Bongen: Open Source Interface Politics: Identity, Acceptance, Trust, and Lobbying. In: *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, ACM, 2011. pp. 1723–1728. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1979835>

[81] O. Badreddin, T.C. Lethbridge, M. Elassar: Modeling Practices in Open Source Software. In: *Open*

Source Software: Quality Verification, Springer, 2013., pp. 127–139. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-38928-3_9>

[82] M. Rashid, P.M. Clarke, R.V. O'Connor: Exploring Knowledge Loss in Open Source Software (OSS) Projects. In: *Software Process Improvement and Capability Determination*, Springer, Cham, 2017 október 4. pp. 481–495.

[83] P.M. Bach, R. DeLine, J.M. Carroll: Designers Wanted: Participation and the User Experience in Open Source Software Development. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2009. pp. 985–994. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1518852>

[84] D. Pagano, W. Maalej: How do open source communities blog? In: *Empirical Software Engineering*, 2013 december. Vol. 18, no. 6, pp. 1090–1124, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-012-9211-2.

[85] T. Otte, R. Moreton, H.D. Knoell: Applied Quality Assurance Methods under the Open Source Development Model., IEEE, 2008. pp. 1247–1252.

[86] Adanna Ezeala, Hyunju Kim, Loretta A. Moore: Open source software development: Expectations and experience from a small development project. In: *Proceedings of the 46th Annual Southeast Regional Conference on XX*, New York, N.Y.: ACM Press, 2008.

[87] W. Kim, S. Chung, B. Endicott-Popovsky: Software architecture model driven reverse engineering approach to open source software development., ACM Press, 2014. pp. 9–14.

[88] S.A. Mokhov, M.-A. Laverdière, D. Benredjem: Taxonomy of Linux Kernel Vulnerability Solutions. in: ISKANDER, Magued (szerk.), In: *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*, Springer Netherlands, 2008., pp. 485–493. ISBN 978-1-4020-8738-7 978-1-4020-8739-4.

[89] P.B. Laat: How can contributors to open-source communities be trusted? On the assumption, inference, and substitution of trust. In: *Ethics and Information Technology*, 2010 december. Vol. 12, no. 4, pp. 327–341, ISSN 1388-1957, 1572-8439. DOI 10.1007/s10676-010-9230-x.

[90] P. Abate, R. Di Cosmo, R. Treinen, S. Zacchiroli: Dependency solving: A separate concern in com-

ponent evolution management. In: *Journal of Systems and Software*, 2012 október. Vol. 85, no. 10, pp. 2228–2240, ISSN 01641212. DOI 10.1016/j.jss.2012.02.018.

[91] A. Ampatzoglou, S. Charalampidou, I. Stamelos: Investigating the Use of Object-Oriented Design Patterns in Open-Source Software: A Case Study. In: *Evaluation of Novel Approaches to Software Engineering*, Springer, 2011., pp. 106–120. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-23391-3_8>

[92] S.Y. Sohn, M.S. Mok: A strategic analysis for successful open source software utilization based on a structural equation model. In: *Journal of Systems and Software*, 2008 június. Vol. 81, no. 6, pp. 1014–1024, ISSN 01641212. DOI 10.1016/j.jss.2007.08.034.

[93] K. Luther, K. Caine, K. Ziegler, A. Bruckman: Why It Works (When It Works): Success Factors in Online Creative Collaboration. In: *Proceedings of the 16th ACM International Conference on Supporting Group Work*, New York, NY, USA: ACM, 2010. pp. 1–10.

[94] H. Orsila, J. Geldenhuys, A. Ruokonen, I. Hammouda: Update Propagation Practices in Highly Reusable Open Source Components. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 159–170. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_13>

[95] R. Milev, S. Muegge, M. Weiss: Design Evolution of an Open Source Project Using an Improved Modularity Metric. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 20–33. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_4>

[96] I. Ahmed, S. Ghorashi, C. Jensen: An Exploration of Code Quality in FOSS Projects. In: *Open Source Software: Mobile Open Source Technologies*, Springer, 2014., pp. 181–190. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-55128-4_26>

[97] B. Adams, R. Kavanagh, A.E. Hassan, D.M. German: An empirical study of integration activities in distributions of open source software. In: *Empirical Software Engineering*, 2015 március 31. ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-015-9371-y.

[98] K.-J. Stol, M.A. Babar, P. Avgeriou, B. Fitzgerald: A comparative study of challenges in integrating Open Source Software and Inner Source Software. In: *Information and Software Technology*, 2011

december. Vol. 53, no. 12, pp. 1319–1336, ISSN 09505849. DOI 10.1016/j.infsof.2011.06.007.

[99] P. Abate, R.D. Cosmo: Adoption of Academic Tools in Open Source Communities: The Debian Case Study. In: *Open Source Systems: Towards Robust Practices*, Springer, Cham, 2017 május 22. pp. 139–150.

[100] A. Zaimi, A. Ampatzoglou, N. Triantafyllidou, A. Chatzigeorgiou, A. Mavridis, T. Chaikalis, I. Deligiannis, P. Sfetsos, I. Stamelos: An Empirical Study on the Reuse of Third-Party Libraries in Open-Source Software Development., ACM Press, 2015. pp. 1–8.

[101] D. Ahmad: Two Years of Broken Crypto: Debian's Dress Rehearsal for a Global PKI Compromise. In: *Security & Privacy, IEEE*, 2008. Vol. 6, no. 5, pp. 70–73, [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4639029>

[102] N.D. Anh, D.S. Cruzes, R. Conradi, M. Höst, X. Franch, C. Ayala: Collaborative Resolution of Requirements Mismatches When Adopting Open Source Components. In: *Requirements Engineering: Foundation for Software Quality*, Springer, 2012., pp. 77–93. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-28714-5_7>

[103] C.A. Ardagna, M. Banzi, E. Damiani, F. Frati: Assurance Process for Large Open Source Code Bases., IEEE, 2009. pp. 412–417.

[104] A. De Groot, S. Kügler, P.J. Adams, G. Gousios: Call for Quality: Open Source Software Quality Observation. In: *Open Source Systems*, Springer, 2006., pp. 57–62. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/0-387-34226-5_6>

[105] Y. Kuwata, K. Takeda, H. Miura: A Study on Maturity Model of Open Source Software Community to Estimate the Quality of Products. In: *Procedia Computer Science*, 2014. Vol. 35, pp. 1711–1717, ISSN 18770509. DOI 10.1016/j.procs.2014.08.264.

[106] A. Bosu, J.C. Carver, M. Hafiz, P. Hilley, D. Janni: Identifying the Characteristics of Vulnerable Code Changes: An Empirical Study. In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA: ACM, 2014. pp. 257–268.

[107] T. Aaltonen, J. Jokinen: Influence in the Linux Kernel Community. In: *Open Source Development,*

Adoption and Innovation, Springer, 2007., pp. 203–208. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_16>

[108] C. Bouras, V. Kokkinos, G. Tseliou: Methodology for Public Administrators for selecting between open source and proprietary software. In: *Telematics and Informatics*, 2013 május. Vol. 30, no. 2, pp. 100–110, ISSN 07365853. DOI 10.1016/j.tele.2012.03.001.

[109] W.D. Sunindyo, T. Moser, D. Winkler, D. Dhungana: Improving Open Source Software Process Quality Based on Defect Data Mining. In: *Software Quality. Process Automation in Software Development*, Springer, 2012. pp. 84–102. [2015-11-27] Retrieved: <http://link.springer.com/content/pdf/10.1007/978-3-642-27213-4.pdf#page=94>

[110] J. Asundi: The Need for Effort Estimation Models for Open Source Software Projects. In: *Proceedings of the Fifth Workshop on Open Source Software Engineering*, New York, NY, USA: ACM, 2005. pp. 1–3.

[111] P.C. Rigby, C. Bird: Convergent Contemporary Software Peer Review Practices. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, New York, NY, USA: ACM, 2013. pp. 202–212.

[112] B.D. Sethanandha, B. Massey, W. Jones: Managing Open Source Contributions for Software Project Sustainability. In: *Technology Management for Global Economic Growth (PICMET), 2010 Proceedings of PICMET'10:* IEEE, 2010. pp. 1–9. [2015-01-18] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5602062>

[113] J. Wang, J.M. Carroll: Behind Linus's Law: A Preliminary Analysis of Open Source Software Peer Review Practices in Mozilla and Python. In: *Collaboration Technologies and Systems (CTS), 2011 International Conference On*, IEEE, 2011. pp. 117–124. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5928673>

[114] P. Thongtanunam, C. Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida, K.-i. Matsumoto: Who Should Review My Code? A File Location-Based Code-Reviewer Recommendation Approach for Modern Code Review. In: *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference On*, IEEE, 2015. pp. 141–150. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/

xpls/abs_all.jsp?arnumber=7081824>

[115] F. Chen, L. Li, J. Jiang, L. Zhang: Predicting the number of forks for open source software project., ACM Press, 2014. pp. 40–47.

[116] L. Yu, A. Mishra, D. Mishra: An Empirical Study of the Dynamics of GitHub Repository and Its Impact on Distributed Software Development. In: *On the Move to Meaningful Internet Systems: OTM 2014 Workshops*, Springer, 2014. pp. 457–466. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-662-45550-0_46>

[117] A. Bosu, J.C. Carver: Impact of developer reputation on code review outcomes in OSS projects: An empirical investigation., ACM Press, 2014. pp. 1–10.

[118] S. Bouktif, H. Sahraoui, F. Ahmed: Predicting Stability of Open-Source Software Systems Using Combination of Bayesian Classifiers. In: *ACM Transactions on Management Information Systems*, 2014 április 1. Vol. 5, no. 1, pp. 1–26, ISSN 2158656X. DOI 10.1145/2555596.

[119] Y. Brun, R. Holmes, M.D. Ernst, D. Notkin: Early Detection of Collaboration Conflicts and Risks. In: *IEEE Transactions on Software Engineering*, 2013 október. Vol. 39, no. 10, pp. 1358–1375, ISSN 0098-5589, 1939-3520. DOI 10.1109/TSE.2013.28.

[120] L. Nyman: Hackers on Forking., ACM Press, 2014. pp. 1–10.

[121] X. Hao, Z. Zhengang, L. Chunpei, D. Zhuo: The Study on Innovation Mechanism of Open Source Software Community. In: *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference On*, IEEE, 2008. pp. 1–5. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4681217>

[122] L. Nyman, T. Mikkonen, J. Lindman, M. Fougère: Perspectives on Code Forking and Sustainability in Open Source Software. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 274–279. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-33442-9_21>

[123] A. Ihara, A. Monden, K.-I. Matsumoto: Industry Questions about Open Source Software in Business: Research Directions and Potential Answers., IEEE, 2014 november. pp. 55–59.

[124] T. Björgvinsson, H. Thorbergsson: Software Development for Governmental Use Utilizing Free and

Open Source Software. In: *Proceedings of the 1st International Conference on Theory and Practice of Electronic Governance*, ACM, 2007. pp. 133–140. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1328087>

[125] R. Takasawa, K. Sakamoto, A. Ihara, H. Washizaki, Y. Fukazawa: Do Open Source Software Projects Conduct Tests Enough? In: *Product-Focused Software Process Improvement*, Springer, 2014., pp. 322–325. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-319-13835-0_32>

[126] E. Capra, A.I. Wasserman: A Framework for Evaluating Managerial Styles in Open Source Projects. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 1–14. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_1>

[127] A. Gupta, R.K. Singla: An Empirical Investigation of Defect Management in Free/Open Source Software Projects. In: *Advances in Computer and Information Sciences and Engineering*, Springer, 2008., pp. 68–73. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-1-4020-8741-7_13>

[128] D. Tosi, A. Tahir: A Survey on How Well-Known Open Source Software Projects Are Tested. In: *Software and Data Technologies*, Springer, 2013., pp. 42–57. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-29578-2_3>

[129] X. Li, Y.F. Li, M. Xie, S.H. Ng: Reliability analysis and optimal version-updating for open source software. In: *Information and Software Technology*, 2011 szeptember. Vol. 53, no. 9, pp. 929–936, ISSN 09505849. DOI 10.1016/j.infsof.2011.04.005.

[130] S. Dashevskyi, A.D. Brucker, F. Massacci: On the Security Cost of Using a Free and Open Source Component in a Proprietary Product. In: *Engineering Secure Software and Systems*, Springer, Cham, 2016 április 6. pp. 190–206.

[131] J. Teixeira: Release Early, Release Often and Release on Time. An Empirical Case Study of Release Management. In: *Open Source Systems: Towards Robust Practices*, Springer, Cham, 2017 május 22. pp. 167–181.

[132] C. Francalanci, F. Merlo: Empirical Analysis of the Bug Fixing Process in Open Source Projects. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 187–196. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_15>

[133] M. Michlmayr, F. Hunt, D. Probert: Release Management in Free Software Projects: Practices and Problems. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 295–300. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_31>

[134] V. Midha, A. Bhattacherjee: Governance practices and software maintenance: A study of open source projects. In: *Decision Support Systems*, 2012 december. Vol. 54, no. 1, pp. 23–32, ISSN 01679236. DOI 10.1016/j.dss.2012.03.002.

[135] E. Petrinja, A. Sillitti, G. Succi: Overview on Trust in Large FLOSS Communities. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 47–56. [2015-10-21] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_4>

[136] A. Matos, M.P. De Leon, R. Ferreira, J.P. Barraca: An Open Source Software Forge for European Projects. In: *Proceedings of the Workshop on Open Source and Design of Communication*, ACM, 2013. pp. 41–45. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2503857>

[137] B. De Alwis, J. Sillito: Why Are Software Projects Moving from Centralized to Decentralized Version Control Systems? In: *Cooperative and Human Aspects on Software Engineering, 2009. CHASE'09. ICSE Workshop On*, 2009. pp. 36–39. [2013-06-28] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5071408>

[138] M. Squire, A.K. Smith: The Diffusion of Pastebin Tools to Enhance Communication in FLOSS Mailing Lists. in: DAMIANI, Ernesto, FRATI, Fulvio, RIEHLE, Dirk és WASSERMAN, Anthony I. (szerk.), In: *Open Source Systems: Adoption and Impact*, Cham: Springer International Publishing, 2015., pp. 45–57. ISBN 978-3-319-17836-3 978-3-319-17837-0.

[139] P. Di Giacomo: COTS and Open Source Software Components: Are They Really Different on the Battlefield? In: *COTS-Based Software Systems*, Springer, 2005., pp. 301–310. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-30587-3_39>

[140] L. Ramanathan, S.K. Iyer: A Qualitative Study on the Adoption of Open Source Software in Information Technology Outsourcing Organizations. in: DAMIANI, Ernesto, FRATI, Fulvio, RIEHLE, Dirk és WASSERMAN, Anthony I. (szerk.), In: *Open Source Systems: Adoption and Impact*, Cham: Springer International Publishing, 2015., pp. 103–113. ISBN 978-3-319-17836-3 978-3-319-17837-0.

[141] F. Almeida, J. Cruz: Open Source Unified Communications: The New Paradigm to Cut Costs and Extend Productivity. In: *Proceedings of the Workshop on Open Source and Design of Communication*, ACM, 2012. pp. 11–12. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2316939>

[142] T. Noda, T. Tansho, S. Coughlan: Effect on Business Growth by Utilization and Contribution of Open Source Software in Japanese IT Companies. In: *Open Source Software: Quality Verification*, Springer, 2013., pp. 222–231. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-38928-3_16>

[143] J. Marsan, G. Paré: Antecedents of open source software adoption in health care organizations: A qualitative survey of experts in Canada. In: *International Journal of Medical Informatics*, 2013 augusztus. Vol. 82, no. 8, pp. 731–741, ISSN 13865056. DOI 10.1016/j.ijmedinf.2013.04.001.

[144] G. Hofmann, D. Riehle, C. Kolassa, W. Mauerer: A Dual Model of Open Source License Growth. In: *Open Source Software: Quality Verification*, Springer, 2013., pp. 245–256. ISBN 978-3-642-38928-3.

[145] T. Wasserman, E. Capra: Evaluating Software Engineering Processes in Commercial and Community Open Source Projects., 2007. [2015-11-24] Retrieved: <http://repository.cmu.edu/silicon_valley/36/>

[146] J. Lindman, R. Rajala, M. Rossi: FLOSS-Induced Changes in the Software Business: Insights from the Pioneers. In: *Software Business*, Springer, 2010., pp. 199–204. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13633-7_20>

[147] F. Van der Linden: Open Source Practices in Software Product Line Engineering. In: *Software Engineering*, Springer, 2013., pp. 216–235. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-36054-1_8>

[148] J. Asundi, O. Carare, K. Dogan: Competitive implications of software open-sourcing. In: *Decision Support Systems*, 2012 december. Vol. 54, no. 1, pp. 153–163, ISSN 01679236. DOI 10.1016/j.dss.2012.05.001.

[149] Jing Yang, Jiang Wang: Review on Free and Open Source Software., Beijing: IEEE, 2008., pp. 1044–1049. ISBN 978-1-4244-2013-1.

[150] T. Pykäläinen, D. Yang, T. Fang: Alleviating piracy through open source strategy: An exploratory study of business software firms in China. In: *The Journal of Strategic Information Systems*, 2009 december. Vol. 18, no. 4, pp. 165–177, ISSN 09638687. DOI 10.1016/j.jsis.2009.10.001.

[151] H. Thorbergsson, T. Björgvinsson, Á. Valfells: Economic Benefits of Free and Open Source Software in Electronic Governance. In: *Proceedings of the 1st International Conference on Theory and Practice of Electronic Governance*, New York, NY, USA: ACM, 2007. pp. 183–186.

[152] S. Chopra, S. Dexter: Free software and the economics of information justice. In: *Ethics and Information Technology*, 2011 szeptember. Vol. 13, no. 3, pp. 173–184, ISSN 1388-1957, 1572-8439. DOI 10.1007/s10676-010-9226-6.

[153] D.G. Hendry: Public participation in proprietary software development through user roles and discourse. In: *International Journal of Human-Computer Studies*, 2008 július. Vol. 66, no. 7, pp. 545–557, ISSN 10715819. DOI 10.1016/j.ijhcs.2007.12.002.

[154] A. Tesoriere, L. Balletta: A Dynamic Model of Open Source vs Proprietary R&D. In: *European Economic Review*, 2017 május 1. Vol. 94, pp. 221–239, ISSN 0014-2921. DOI 10.1016/j.euroecorev.2017.02.009.

[155] K. Tanihana, T. Noda: Empirical Study of the Relation between Open Source Software Use and Productivity of Japan's Information Service Industries. In: *Open Source Software: Quality Verification*, Springer, 2013., pp. 18–29. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-38928-3_2>

[156] F. Rentocchini, G. De Prato: Software Patents and Open Source Software in the European Union: Evidences of a Trade-Off? In: *Open Source Systems*, Springer, 2006., pp. 349–351. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/0-387-34226-5_41>

[157] Zhang Yingkui, Zhang Jing, Wang Liye: Justification of Free Software and Its Enlightenment., IEEE, 2010 december. pp. 171–173.

[158] L. Lin: Impact of user skills and network effects on the competition between open source and proprietary software. In: *Electronic Commerce Research and Applications*, 2008 március. Vol. 7, no. 1, pp. 68–81, ISSN 15674223. DOI 10.1016/j.elerap.2007.01.003.

[159] R. Anderson, T. Moore: Information Security Economics–and Beyond. In: *Advances in Cryptology-CRYPTO 2007*, Springer, 2007., pp. 68–91. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-74143-5_5>

[160] J. Marsan, G. Paré, A. Beaudry: Adoption of open source software in organizations: A socio-cognitive perspective. In: *The Journal of Strategic Information Systems*, 2012 december. Vol. 21, no. 4, pp. 257–273, ISSN 09638687. DOI 10.1016/j.jsis.2012.05.004.

[161] A. Barua, S.W. Thomas, A.E. Hassan: What are developers talking about? An analysis of topics and trends in Stack Overflow. In: *Empirical Software Engineering*, 2014 június. Vol. 19, no. 3, pp. 619–654, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-012-9231-y.

[162] Ø. Hauge, D.S. Cruzes, R. Conradi, K.S. Velle, T.A. Skarpenes: Risks and Risk Mitigation in Open Source Software Adoption: Bridging the Gap between Literature and Practice. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 105–118. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_9>

[163] A. Stefi, M. Berger, T. Hess: What Influences Platform Provider's Degree of Openness?–Measuring and Analyzing the Degree of Platform Openness. In: *Software Business. Towards Continuous Value Delivery*, Springer, 2014., pp. 258–272. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-319-08738-2_18>

[164] M. Di Penta, D.M. German, Y.-G. Guéhéneuc, G. Antoniol: An Exploratory Study of the Evolution of Software Licensing. In: *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, New York, NY, USA: ACM, 2010. pp. 145–154.

[165] K.L. Gwebu, J. Wang: Adoption of Open Source Software: The role of social identification. In: *Decision Support Systems*, 2011 április. Vol. 51, no. 1, pp. 220–229, ISSN 01679236. DOI 10.1016/j.dss.2010.12.010.

[166] D. Brink, L. Roos, J. Weller, J.-P. Van Belle: Critical Success Factors for Migrating to OSS-on-the-Desktop: Common Themes across Three South African Case Studies. In: *Open Source Systems*, Springer, 2006., pp. 287–293. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/0-387-34226-5_29>

[167] D. Durand, J.-L. Vuattoux, P.-J. Ditscheid: OSS in 2012: A Long-Term Sustainable Alternative for Corporate IT. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 371–376. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-33442-9_37>

[168] K. Ven, J. Verelst: A Field Study on the Barriers in the Assimilation of Open Source Server Software. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 281–293. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_22>

[169] S. Goode: Something for nothing: Management rejection of open source software in Australia's top firms. In: *Information & Management*, 2005 július. Vol. 42, no. 5, pp. 669–681, ISSN 03787206. DOI 10.1016/j.im.2004.01.011.

[170] C.A. Ardagna, E. Damiani, F. Frati: FOCSE: An OWA-Based Evaluation Framework for OS Adoption in Critical Environments. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 3–16. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_1>

[171] B. Fitzgerald, K.-J. Stol, S. Minör, H. Cosmo: The journeys. In: *Scaling a Software Business*, Springer, Cham, 2017., pp. 56–235. ISBN 978-3-319-53115-1 978-3-319-53116-8.

[172] V. Kuechler, C. Jensen, D. Bryant: Misconceptions and Barriers to Adoption of FOSS in the US Energy Industry. In: *Open Source Software: Quality Verification*, Springer, 2013., pp. 232–244. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-38928-3_17>

[173] M. Shaikh, T. Cornford: Framing the Conundrum of Total Cost of Ownership of Open Source Software. In: *Open Source Systems: Grounding Research*, Springer, 2011., pp. 208–219. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-24418-6_14>

[174] M. Dolores Gallego, S. Bueno, F. José Racero, J. Noyes: Open source software: The effects of training on acceptance. In: *Computers in Human Behavior*, 2015 augusztus. Vol. 49, pp. 390–399, ISSN 07475632. DOI 10.1016/j.chb.2015.03.029.

[175] N. Lazić, M.B. Zorica, J. Klindžić: Open Source Software in Education. In: *MIPRO, 2011 Proceedings of the 34th International Convention*, IEEE, 2011. pp. 1267–1270. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5967252>

[176] M. Sabin: Free and Open Source Software Development of IT Systems. In: *Proceedings of the 2011 Conference on Information Technology Education*, New York, NY, USA: ACM, 2011. pp. 27–32.

[177] A. Barcomb, M. Grottke, J.-P. Stauffert, D. Riehle, S. Jahn: How Developers Acquire FLOSS Skills. In: *Open Source Systems: Adoption and Impact*, Springer, Cham, 2015 május 16. pp. 23–32.

[178] M. Ciolkowski, M. Soto: Towards a Comprehensive Approach for Assessing Open Source Projects. In: *Software Process and Product Measurement*, Springer, 2008., pp. 316–330. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-89403-2_26>

[179] D. Taibi, L. Lavazza, S. Morasca: OpenBQR: A Framework for the Assessment of OSS. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 173–186. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_14>

[180] Y. Zhou, J. Davis: Open Source Software Reliability Model: An Empirical Approach. In: *ACM SIGSOFT Software Engineering Notes*, ACM, 2005. pp. 1–6. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1083273>

[181] M. Höst, A. Oručević-Alagić, P. Runeson: Usage of Open Source in Commercial Software Product Development–Findings from a Focus Group Meeting. In: *Product-Focused Software Process Improvement*, Springer, 2011., 6759. pp. 143–155. ISBN 978-3-642-21843-9.

[182] J. Gamalielsson, B. Lundell: Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? In: *Journal of Systems and Software*, 2014 március. Vol. 89, pp. 128–145, ISSN 01641212. DOI 10.1016/j.jss.2013.11.1077.

[183] N. Ullah: A method for predicting open source software residual defects. In: *Software Quality Journal*, 2014 április 8. ISSN 0963-9314, 1573-1367. DOI 10.1007/s11219-014-9229-3.

[184] M. Morandini, A. Siena, A. Susi: Risk Awareness in Open Source Component Selection. In: *Business Information Systems*, Springer, 2014. pp. 241–252. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-319-06695-0_21>

[185] K.-J. Stol, M. Ali Babar: Challenges in Using Open Source Software in Product Development: A Review of the Literature. In: *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, ACM, 2010. pp. 17–22. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1833276>

[186] Simon Harrer, Jörg Lenhard, Guido Wirtz: Open Source versus Proprietary Software in Service-Orientation: The Case of BPEL Engines. In: *Open Source versus Proprietary Software in Service-Orientation: The Case of BPEL Engines*, Berlin, Heidelberg: Springer, 2013., pp. 99–113. ISBN 978-3-

642-45005-1.

[187] C. Ruiz, W. Robinson: Towards a Unified Definition of Open Source Quality. In: *Open Source Systems: Grounding Research*, Springer, 2011., pp. 17–33. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-24418-6_2>

[188] E. Petrinja, G. Succi: Two Evolution Indicators for FOSS Projects. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 216–232. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-33442-9_14>

[189] D. Taibi, V. Del Bianco, D. Dalle Carbonare, L. Lavazza, S. Morasca: Towards The Evaluation of OSS Trustworthiness: Lessons Learned From The Observation of Relevant OSS Projects. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 389–395. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_37>

[190] European Comission: FP6-033982: *FLOSSMetrics Final Report*. European Comission, 2010. [2012-10-20] Retrieved: <http://flossmetrics.org/docs/fm3-final-report_en.pdf>

[191] A.I. Wasserman, X. Guo, B. McMillian, K. Qian, M.-Y. Wei, Q. Xu: OSSpal: Finding and Evaluating Open Source Software. In: *Open Source Systems: Towards Robust Practices*, Springer, Cham, 2017 május 22. pp. 193–203.

[192] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P.J. Adams, I. Samoladas, I. Stamelos: Evaluating the Quality of Open Source Software. In: *Electronic Notes in Theoretical Computer Science*, 2009 március. Vol. 233, pp. 5–28, ISSN 15710661. DOI 10.1016/j.entcs.2009.02.058.

[193] J.-C. Deprez, S. Alexandre: Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS. In: *Product-Focused Software Process Improvement*, Springer, 2008., pp. 189–203. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-69566-0_17>

[194] E. Petrinja, A. Sillitti, G. Succi: Comparing OpenBRR, QSOS, and OMM Assessment Models. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 224–238. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_18>

[195] A. Adewumi, S. Misra, N. Omoregbe: A Review of Models for Evaluating Quality in Open Source Software. In: *IERI Procedia*, 2013. Vol. 4, pp. 88–92, ISSN 22126678. DOI 10.1016/j.ieri.2013.11.014.

[196] I. Samoladas, G. Gousios, D. Spinellis, I. Stamelos: The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 237–248. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_19>

[197] Umm-e-Laila, A. Zahoor, K. Mehboob, S. Natha: Comparison of Open Source Maturity Models. In: *Procedia Computer Science*, 2017 január 1. Vol. 111, pp. 348–354, ISSN 1877-0509. DOI 10.1016/j.procs.2017.06.033.

[198] O. Franco-Bedoya, D. Ameller, D. Costal, X. Franch: Measuring the Quality of Open Source Software Ecosystems Using QuESo. In: *Software Technologies*, Springer, Cham, 2014 augusztus 29. pp. 39–62.

[199] A. Kritikos, I. Stamelos: Open Source Software Resilience Framework. In: *Open Source Systems: Enterprise Software and Solutions*, Springer, Cham, 2018 június 8. pp. 39–49.

[200] F.-B. Oscar, D. Ameller, D. Costal, X. Franch: QuESo: A Quality Model for Open Source Software Ecosystems. In: *ICSOFT-EA 2014 - Proceedings of the 9th International Conference on Software Engineering and Applications*, 2014 augusztus 28.

[201] A. Siena, M. Morandini, A. Susi: Modelling Risks in Open Source Software Component Selection. In: *Conceptual Modeling*, Springer, 2014., pp. 335–348. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-319-12206-9_28>

[202] C.D. Jensen, M.B. Nielsen: CodeTrust. In: *Trust Management XII*, Springer, Cham, 2018 július 9. pp. 58–74.

[203] R.Y. Arakji, K.R. Lang: The Virtual Cathedral and the Virtual Bazaar. In: *ACM SIGMIS Database*, 2007. Vol. 38, no. 4, pp. 32–39, [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1314242>

[204] V. Dilmurad, E. Kerem, Ç. Murat, S.B. Sami: Open source software usage on municipalities; a case study: Çankaya municipality. In: *Procedia Computer Science*, 2011. Vol. 3, pp. 805–808, ISSN 18770509. DOI 10.1016/j.procs.2010.12.132.

[205] C. De Pablos, D. López: The Adoption of Open Source Systems in Public Administrations. In:

ENTERprise Information Systems, Springer, 2010., pp. 138–146. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-16402-6_15>

[206] C. Ayala, X. Franch, R. Conradi, J. Li, D. Cruzes: Developing Software with Open Source Software Components. In: *Finding Source Code on the Web for Remix and Reuse*, Springer, 2013., pp. 167–186. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-1-4614-6596-6_9>

[207] M. Silic: Dual-use open source security software in organizations – Dilemma: Help or hinder? In: *Computers & Security*, 2013 november. Vol. 39, pp. 386–395, ISSN 01674048. DOI 10.1016/j.cose.2013.09.003.

[208] W. Chen, J. Li, J. Ma, R. Conradi, J. Ji, C. Liu: A Survey of Software Development with Open Source Components in Chinese Software Industry. In: *Software Process Dynamics and Agility*, Springer, 2007., pp. 208–220. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-72426-1_18>

[209] M. Huang, L. Yang, Y. Yang: A Development Process for Building OSS-Based Applications. In: *Unifying the Software Process Spectrum*, Springer, 2006., pp. 122–135. [2015-11-26] Retrieved: <http://link.springer.com/chapter/10.1007/11608035_13>

[210] C. Ayala, A. Nguyen-Duc, X. Franch, M. Höst, R. Conradi, D. Cruzes, M.A. Babar: System requirements-OSS components: Matching and mismatch resolution practices – an empirical study. In: *Empirical Software Engineering*, 2018 március 2. pp. 1–56, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-017-9594-1.

[211] C. Ayala, Ø. Hauge, R. Conradi, X. Franch, J. Li, K.S. Velle: Challenges of the Open Source Component Marketplace in the Industry. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 213–224. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_19>

[212] J. Merilinna, M. Matinlassi: State of the Art and Practice of Opensource Component Integration. In: *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference On*, IEEE, 2006. pp. 170–177. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1690138>

[213] Krzysztof Wnuk, David Callele, Even-André Karlsson, Dietmar Pfahl: Value-Based Software Engi-

neering. In: *ESEM '12*, Berlin: Springer, 2012. pp. 271–280.

[214] T. Nishimura, H. Sato: Analysis of a Security Incident of Open Source Middleware – Case Analysis of 2008 Debian Incident of OpenSSL., IEEE, 2009 július. pp. 247–250.

[215] M. Rajanen, N. Iivari, A. Lanamäki: Non-response, Social Exclusion, and False Acceptance: Gatekeeping Tactics and Usability Work in Free-Libre Open Source Software Development. In: *Human-Computer Interaction – INTERACT 2015*, Springer, Cham, 2015 szeptember 14. pp. 9–26.

[216] N. Harutyunyan, A. Bauer, D. Riehle: Understanding Industry Requirements for FLOSS Governance Tools. In: *Open Source Systems: Enterprise Software and Solutions*, Springer, Cham, 2018 június 8. pp. 151–167.

[217] N. Nissanke: Component Security-Issues and an Approach. In: *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, IEEE, 2005. pp. 152–155. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1508103>

[218] T. Yamakami: Challenges for Mobile Middleware Platform: Issues for Embedded Open Source Software Integration. In: *OSS*, Springer, 2010. pp. 401–406. [2015-11-27] Retrieved: <http://link.springer.com/content/pdf/10.1007/978-3-642-13244-5.pdf#page=418>

[219] Budai Balázs Benjámin: *E-Közigazgatás Axiomatikus Megközelítésben*. Doktori értekezés. Pécs: Pécsi Tudományegyetem Állam- és Jogtudományi Kar, 2008.

[220] E. Luoma, N. Helander, L. Frank: Adoption of Open Source Software and Software-as-a-Service Models in the Telecommunication Industry. In: *Software Business*, Springer, 2011., pp. 70–84. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-21544-5_7>

[221] M. Välimäki, V. Oksanen, J. Laine: An Empirical Look at the Problems of Open Source Adoption in Finnish Municipalities. In: *Proceedings of the 7th International Conference on Electronic Commerce*, New York, NY, USA: ACM, 2005. pp. 514–520.

[222] G. Cheung, P. Chilana, S. Kane, B. Pellett: Designing for Discovery: Opening the Hood for Open-Source End User Tinkering. In: *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, ACM, 2009. pp. 4321–4326. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1520660>

[223] K.L. Gwebu, J. Wang: Seeing eye to eye? An exploratory study of free open source software users' perceptions. In: *Journal of Systems and Software*, 2010 november. Vol. 83, no. 11, pp. 2287–2296, ISSN 01641212. DOI 10.1016/j.jss.2010.07.011.

[224] M. Silic, A. Back: Information Security and Open Source Dual Use Security Software: Trust Paradox. In: *Open Source Software: Quality Verification*, Springer, 2013., pp. 194–206. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-38928-3_14>

[225] Y. Li, C.-H. Tan, X. Yang: It is all about what we have: A discriminant analysis of organizations' decision to adopt open source software. In: *Decision Support Systems*, 2013 december. Vol. 56, pp. 56–62, ISSN 01679236. DOI 10.1016/j.dss.2013.05.006.

[226] A. Mahajan, B. Clarysse: Technological Innovation and Resource Bricolage in Firms: The Role of Open Source Software. In: *Open Source Software: Quality Verification*, Springer, 2013., pp. 1–17. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-38928-3_1>

[227] W. Scacchi: *Understanding Requirements for Open Source Software*, S.l.: Springer, 2009. [2015-10-21] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-92966-6_27>

[228] M.C. Jaeger: Open Source Issues with Cloud Storage Software. In: *Advances in Service-Oriented and Cloud Computing*, Springer, 2013., pp. 106–113. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-45364-9_10>

[229] E. Luoma, M. Riepula, L. Frank: Scenarios on Adoption of Open Source Software in the Communications Software Industry. In: *Software Business*, Springer, 2011., pp. 110–124. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-21544-5_10>

[230] S. Mansfield-Devine: Open Source Software: Determining the Real Risk Posed by Vulnerabilities. In: *Network Security*, 2017 január 1. Vol. 2017, no. 1, pp. 7–12, ISSN 1353-4858. DOI 10.1016/S1353-4858(17)30005-3.

[231] R. Viseur: Initial Results from the Study of the Open Source Sector in Belgium., ACM Press, 2014. pp. 1–5.

[232] E. Diehl: Law 3: No Security Through Obscurity. In: *Ten Laws for Security*, Springer, Cham, 2016., pp. 67–79. ISBN 978-3-319-42639-6 978-3-319-42641-9.

[233] G. László: *A Nyílt Forráskódú Szoftverek Társadalmi-Gazdasági Hatásainak Feltárása a Központi Kezdeményezések Tükrében*. 2009. Retrieved: <http://www.omikk.bme.hu/collections/phd/Gazdasag_es_Tarsadalomtudomanyi_Kar/2009/Laszlo_Gabor/ertekezes.pdf>

[234] J. Akkanen, H. Demeter, T. Eppel, Z. Iványfi, J.K. Nurminen, P. Stenman: Reusing an Open Source Application—Practical Experiences with a Mobile CRM Pilot. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 217–222. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_18>

[235] S. Ke\s sler, P. Alpar: Customization of Open Source Software in Companies. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 129–142. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_13>

[236] L. Morgan, P. Finnegan: Benefits and Drawbacks of Open Source Software: An Exploratory Study of Secondary Software Firms. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 307–312. ISBN 978-0-387-72486-7.

[237] L. Morgan, P. Finnegan: Beyond free software: An exploration of the business value of strategic open source. In: *The Journal of Strategic Information Systems*, 2014 szeptember. Vol. 23, no. 3, pp. 226–238, ISSN 09638687. DOI 10.1016/j.jsis.2014.07.001.

[238] J. Recker, M. La Rosa: Understanding user differences in open-source workflow management system usage intentions. In: *Information Systems*, 2012 május. Vol. 37, no. 3, pp. 200–212, ISSN 03064379. DOI 10.1016/j.is.2011.10.002.

[239] D. Tosi, L. Lavazza, S. Morasca, M. Chiappa: Surveying the Adoption of FLOSS by Public Administration Local Organizations. in: DAMIANI, Ernesto, FRATI, Fulvio, RIEHLE, Dirk és WASSERMAN, Anthony I. (szerk.), In: *Open Source Systems: Adoption and Impact*, Cham: Springer International Publishing, 2015., pp. 114–123. ISBN 978-3-319-17836-3 978-3-319-17837-0.

[240] L.A. Joia, J.C. Dos Santos Vinhais: From Closed Source to Open Source Software: Analysis of the Migration Process to Open Office. In: *The Journal of High Technology Management Research*, 2017 január 1. Vol. 28, no. 2, pp. 261–272, ISSN 1047-8310. DOI 10.1016/j.hitech.2017.10.008.

[241] K.-J. Stol, M.A. Babar, P. Avgeriou: The Importance of Architectural Knowledge in Integrating Open

Source Software. In: *Open Source Systems: Grounding Research*, Springer, 2011., pp. 142–158. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-24418-6_10>

[242] M. Vouk, L. Williams, Others: Using Software Reliability Models for Security Assessment—Verification of Assumptions. In: *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium On*, IEEE, 2013. pp. 23–24. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6688858>

[243] S. Zanero, E. Huebner: The Case for Open Source Software in Digital Forensics. in: HUEBNER, Ewa és ZANERO, Stefano (szerk.), In: *Open Source Software for Digital Forensics*, Boston, MA: Springer US, 2010., pp. 3–7. ISBN 978-1-4419-5802-0 978-1-4419-5803-7.

[244] B. Lundell, B. Lings, A. Syberfeldt: Practitioner perceptions of Open Source software in the embedded systems area. In: *Journal of Systems and Software*, 2011 szeptember. Vol. 84, no. 9, pp. 1540–1549, ISSN 01641212. DOI 10.1016/j.jss.2011.03.020.

[245] D. Cerri, A. Fuggetta: Open standards, open formats, and open source. In: *Journal of Systems and Software*, 2007 november. Vol. 80, no. 11, pp. 1930–1937, ISSN 01641212. DOI 10.1016/j.jss.2007.01.048.

[246] T. Huynh, J. Miller: An empirical investigation into open source web applications' implementation vulnerabilities. In: *Empirical Software Engineering*, 2010 október. Vol. 15, no. 5, pp. 556–576, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-010-9131-y.

[247] K. Achuthan, S. SudhaRavi, R. Kumar, R. Raman: Security Vulnerabilities in Open Source Projects: An India Perspective. In: *Information and Communication Technology (ICoICT), 2014 2nd International Conference On*, IEEE, 2014. pp. 18–23. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6914033>

[248] W. Westerhof: Theoretical Concept – Horus Scenario.. 2017. [2018-09-17] Retrieved: <https://horusscenario.com/theoretical-concept/>

[249] SMA: Whitepaper Cyber Security., 2017. pp. 9,

[250] A. Khelifi, M. Aburrous, M.A. Talib, P. Shastry: Enhancing Protection Techniques of E-Banking Security Services Using Open Source Cryptographic Algorithms., IEEE, 2013 július. pp. 89–95.

[251] O. Alhazmi, Y. Malaiya, I. Ray: Security Vulnerabilities in Software Systems: A Quantitative Perspective. In: *Data and Applications Security XIX*, Springer, 2005., pp. 281–294. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/11535706_21>

[252] J. Lindman, I. Hammouda: Support mechanisms provided by FLOSS foundations and other entities. In: *Journal of Internet Services and Applications*, 2018 december 1. Vol. 9, no. 1, pp. 8, ISSN 1867-4828, 1869-0238. DOI 10.1186/s13174-018-0079-2.

[253] M. Banzi, G. Bruno, G. Caire: To What Extent Does It Pay to Approach Open Source Software for a Big Telco Player? In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 307–315. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_27>

[254] C. Bayrak, C. Davis: Open Source Software Development: Structural Tension in the American Experiment. In: *Advances in Computers*, Elsevier, 2005., pp. 247–282. ISBN 978-0-12-012164-9.

[255] A. Filippova, H. Cho: Mudslinging and Manners: Unpacking Conflict in Free and Open Source Software., ACM Press, 2015. pp. 1393–1403.

[256] E. Di Bella, A. Sillitti, G. Succi: A multivariate classification of open source developers. In: *Information Sciences*, 2013 február. Vol. 221, pp. 72–83, ISSN 00200255. DOI 10.1016/j.ins.2012.09.031.

[257] T. Kilamo, T. Aaltonen, T.J. Heinimäki: BULB: Onion-Based Measuring of OSS Communities. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 342–347. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_29>

[258] S. Toral, M. Martínez-Torres, F. Barrero: Analysis of virtual communities supporting OSS projects using social network analysis. In: *Information and Software Technology*, 2010 március. Vol. 52, no. 3, pp. 296–303, ISSN 09505849. DOI 10.1016/j.infsof.2009.10.007.

[259] I. Scholtes, P. Mavrodiev, F. Schweitzer: From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in Open Source Software projects. In: *Empirical Software Engineering*, 2016 április 1. Vol. 21, no. 2, pp. 642–683, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-015-9406-4.

[260] C. Oezbek, L. Prechelt, F. Thiel: The Onion Has Cancer: Some Social Network Analysis Visualizations of Open Source Project Communication. In: *Proceedings of the 3rd International Workshop*

on Emerging Trends in Free/Libre/Open Source Software Research and Development, New York, NY, USA: ACM, 2010. pp. 5–10.

[261] K. Yamashita, S. McIntosh, Y. Kamei, A.E. Hassan, N. Ubayashi: Revisiting the Applicability of the Pareto Principle to Core Development Teams in Open Source Software Projects. In: *Proceedings of the 14th International Workshop on Principles of Software Evolution*, ACM, 2015. pp. 46–55. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2804366>

[262] F. Ricca, A. Marchetto: Heroes in FLOSS Projects: An Explorative Study., IEEE, 2010 október. pp. 155–159.

[263] D. Garcia, M.S. Zanetti, F. Schweitzer: The Role of Emotions in Contributors Activity: A Case Study on the GENTOO Community., IEEE, 2013 szeptember. pp. 410–417.

[264] Ø. Hauge, C.-F. Sørensen, A. Røsdal: Surveying Industrial Roles in Open Source Software Development. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 259–264. [2014-12-20] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_25>

[265] A. Barham: The Impact of Formal QA Practices on FLOSS Communities–The Case of Mozilla. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 262–267. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-33442-9_19>

[266] C. Bird, D. Pattison, R. D'Souza, V. Filkov, P. Devanbu: Latent Social Structure in Open Source Projects. In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA: ACM, 2008. pp. 24–35.

[267] A. Jermakovics, A. Sillitti, G. Succi: Exploring Collaboration Networks in Open-Source Projects. In: *Open Source Software: Quality Verification*, Springer, 2013. pp. 97–108. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-38928-3_7>

[268] A. Capiluppi, P.J. Adams: Reassessing Brooks' Law for the Free Software Community. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 274–283. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_24>

[269] A. Capiluppi, M. Michlmayr: From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects. In: *Open Source Development, Adoption and Innovation*, Springer,

2007., pp. 31–44. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_3>

[270] D. Forrest, C. Jensen, N. Mohan, J. Davidson: Exploring the Role of Outside Organizations in Free/Open Source Software Projects. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 201–215. ISBN 978-3-642-33442-9.

[271] M. Meike, J. Sametinger, A. Wiesauer: Security in Open Source Web Content Management Systems. In: *IEEE Security & Privacy*, 2009. no. 4, pp. 44–51, [2015-11-24] Retrieved: <http://www.computer.org/web/csdl/index/-/csdl/mags/sp/2009/04/msp2009040044.html>

[272] M. Schaarschmidt, G. Walsh, H.F. Von Kortzfleisch: How do firms influence open source software communities? A framework and empirical analysis of different governance modes. In: *Information and Organization*, 2015 április. Vol. 25, no. 2, pp. 99–114, ISSN 14717727. DOI 10.1016/j.infoandorg.2015.03.001.

[273] M. Liu, J.L. Zhao: On Outsourcing and Offshoring in Community Source. In: *Advanced Management of Information for Globalized Enterprises, 2008. AMIGE 2008. IEEE Symposium On*, IEEE, 2008. pp. 1–3. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4721524>

[274] M. Martínez-Torres: A genetic search of patterns of behaviour in OSS communities. In: *Expert Systems with Applications*, 2012 december. Vol. 39, no. 18, pp. 13182–13192, ISSN 09574174. DOI 10.1016/j.eswa.2012.05.083.

[275] R. Zilouchian Moghaddam, B. Bailey, W.-T. Fu: Consensus Building in Open Source User Interface Design Discussions. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2012. pp. 1491–1500. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2208611>

[276] A. Bosu, J. Carver, R. Guadagno, B. Bassett, D. McCallum, L. Hochstein: Peer impressions in open source organizations: A survey. In: *Journal of Systems and Software*, 2014 augusztus. Vol. 94, pp. 4–15, ISSN 01641212. DOI 10.1016/j.jss.2014.03.061.

[277] B. Vasilescu, V. Filkov, A. Serebrenik: Perceptions of Diversity on Git Hub: A User Survey., IEEE, 2015 május. pp. 50–56.

[278] J. Martinez-Romo, G. Robles, J.M. Gonzalez-Barahona, M. Ortuño-Perez: Using Social Network

Analysis Techniques to Study Collaboration between a FLOSS Community and a Company. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 171–186. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_14>

[279] E.T. Heli Ikonen, Netta Iivari, Henrik Hedberg: *Controlling the Use of Collaboration Tools in Open Source Software Development*, S.l.: s.n., 2010. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1868914>

[280] S.K. Sowe, I. Stamelos, L. Angelis: Understanding knowledge sharing activities in free/open source software projects: An empirical study. In: *Journal of Systems and Software*, 2008 március. Vol. 81, no. 3, pp. 431–446, ISSN 01641212. DOI 10.1016/j.jss.2007.03.086.

[281] M. Squire: Considering the Use of Walled Gardens for FLOSS Project Communication. In: *Open Source Systems: Towards Robust Practices*, Springer, Cham, 2017 május 22. pp. 3–13.

[282] M. Pinzger, H.C. Gall: Dynamic Analysis of Communication and Collaboration in OSS Projects. in: MISTRÍK, Ivan, GRUNDY, John, HOEK, André és WHITEHEAD, Jim (szerk.), In: *Collaborative Software Engineering*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010., pp. 265–284. ISBN 978-3-642-10293-6 978-3-642-10294-3.

[283] P.J. Adams, A. Capiluppi, A. De Groot: Detecting Agility of Open Source Projects through Developer Engagement. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 333–341. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_30>

[284] I. Yahav, R.S. Kenett, X. Bai: Data Driven Testing of Open Source Software. In: *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, Springer, 2014., pp. 309–321. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-662-45231-8_22>

[285] I. Yahav, R.S. Kenett, X. Bai: Risk Based Testing of Open Source Software (OSS)., IEEE, 2014 július. pp. 638–643.

[286] N. Smith, A. Capiluppi, J. Fernández-Ramil: Users and Developers: An Agent-Based Simulation of Open Source Software Evolution. In: *Software Process Change*, Springer, 2006., pp. 286–293. [2015-11-26] Retrieved: <http://link.springer.com/chapter/10.1007/11754305_31>

[287] X. Yang: Social Network Analysis in Open Source Software Peer Review. In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA: ACM, 2014. pp. 820–822.

[288] A. Sureka, A. Goyal, A. Rastogi: Using Social Network Analysis for Mining Collaboration Data in a Defect Tracking System for Risk and Vulnerability Analysis. In: *Proceedings of the 4th India Software Engineering Conference*, New York, NY, USA: ACM, 2011. pp. 195–204.

[289] K. Yamashita, S. McIntosh, Y. Kamei, N. Ubayashi: Magnet or Sticky? An Oss Project-by-Project Typology. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 2014. pp. 344–347. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2597116>

[290] B. Rossi, B. Russo, G. Succi: Analysis of Open Source Software Development Iterations by Means of Burst Detection Techniques. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 83–93. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_9>

[291] F. Fagerholm, A.S. Guinea, J. Münch, J. Borenstein: The Role of Mentoring and Project Characteristics for Onboarding in Open Source Software Projects. In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA: ACM, 2014. pp. 55:1–55:10.

[292] G. Poo-Caamaño, L. Singer, E. Knauss, D.M. German: Herding Cats: A Case Study of Release Management in an Open Collaboration Ecosystem. In: *Open Source Systems: Integrating Communities*, Springer, Cham, 2016 május 30. pp. 147–162.

[293] K. Carillo, S. Huff, B. Chawner: What Makes a Good Contributor? Understanding Contributor Behavior within Large Free/Open Source Software Projects – A Socialization Perspective. In: *The Journal of Strategic Information Systems*, 2017 december 1. Vol. 26, no. 4, pp. 322–359, ISSN 0963-8687. DOI 10.1016/j.jsis.2017.03.001.

[294] A.N. Duc, D.S. Cruzes, C. Ayala, R. Conradi: Impact of Stakeholder Type and Collaboration on Issue Resolution Time in OSS Projects. In: *Open Source Systems: Grounding Research*, Springer, 2011., pp. 1–16. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-24418-6_1>

[295] E. Stiles, X. Cui: Workings of Collective Intelligence within Open Source Communities. In: *Advances in Social Computing*, Springer, 2010., pp. 282–289. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-12079-4_35>

[296] H. Hayashi, A. Ihara, A. Monden, K.-i. Matsumoto: Why is collaboration needed in OSS projects? A case study of eclipse project., ACM Press, 2013. pp. 17–20.

[297] M. AlMarzouq, V. Grover, J.B. Thatcher: Taxing the development structure of open source communities: An information processing view. In: *Decision Support Systems*, 2015 december. Vol. 80, pp. 27–41, ISSN 01679236. DOI 10.1016/j.dss.2015.09.004.

[298] Y. Cai, D. Zhu: Reputation in an Open Source Software Community: Antecedents and Impacts. In: *Decision Support Systems*, 2016 november 1. Vol. 91, pp. 103–112, ISSN 0167-9236. DOI 10.1016/j.dss.2016.08.004.

[299] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, D. Damian: Understanding the popular users: Following, affiliation influence and leadership on GitHub. In: *Information and Software Technology*, 2016 február. Vol. 70, pp. 30–39, ISSN 09505849. DOI 10.1016/j.infsof.2015.10.002.

[300] L. Dahlander, M.W. Wallin: A man on the inside: Unlocking communities as complementary assets. In: *Research Policy*, 2006 október. Vol. 35, no. 8, pp. 1243–1259, ISSN 00487333. DOI 10.1016/j.respol.2006.09.011.

[301] G. Robles, S. Duenas, J.M. Gonzalez-Barahona: Corporate Involvement of Libre Software: Study of Presence in Debian Code over Time. In: *Open Source Development, Adoption and Innovation*, Springer, 2007., pp. 121–132. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-72486-7_10>

[302] G.L. Bruce, J.P. Wittgreffe, J.M.M. Potter, P. Robson: The Potential for Open Source Software in Telecommunications Operational Support Systems. In: *BT technology journal*, 2005. Vol. 23, no. 3, pp. 79–95, [2014-05-31] Retrieved: <http://link.springer.com/article/10.1007/s10550-005-0033-2>

[303] C. Jensen, W. Scacchi: Governance in Open Source Software Development Projects: A Comparative Multi-Level Analysis. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 130–142. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_11>

[304] J. Noll: What Constitutes Open Source? A Study of the Vista Electronic Medical Record Software. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 310–319. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_27>

[305] M. Rajanen, N. Iivari: Open Source and Human Computer Interaction Philosophies in Open Source Projects: Incompatible or Co-Existent? In: *Proceedings of International Conference on Making Sense of Converging Media*, New York, NY, USA: ACM, 2013. pp. 67:67–67:74.

[306] M. Rajanen, N. Iivari: Examining Usability Work and Culture in OSS. in: DAMIANI, Ernesto, FRATI, Fulvio, RIEHLE, Dirk és WASSERMAN, Anthony I. (szerk.), In: *Open Source Systems: Adoption and Impact*, Cham: Springer International Publishing, 2015., pp. 58–67. ISBN 978-3-319-17836-3 978-3-319-17837-0.

[307] J. Järvensivu, T. Mikkonen: Forging a Community–Not: Experiences on Establishing an Open Source Project. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 15–27. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_2>

[308] A. Johri, O. Nov, R. Mitra: ”Cool” or ”Monster”?: Company Takeovers and Their Effect on Open Source Community Participation. In: *Proceedings of the 2011 iConference*, New York, NY, USA: ACM, 2011. pp. 327–331.

[309] L. López, D. Costal, C.P. Ayala, X. Franch, R. Glott, K. Haaland: Modelling and Applying OSS Adoption Strategies. In: *Conceptual Modeling*, Springer, 2014., pp. 349–362. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-319-12206-9_29>

[310] G. Pinto, I. Steinmacher, L.F. Dias, M. Gerosa: On the challenges of open-sourcing proprietary software projects. In: *Empirical Software Engineering*, 2018 március 9. pp. 1–27, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-018-9609-6.

[311] J.-L. Hardy: Industry Regulation through Open Source Software: A Strategic Ownership Proposal. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 322–329. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_26>

[312] J. Teixeira: Understanding Coopetition in the Open-Source Arena: The Cases of WebKit and OpenStack., ACM Press, 2014. pp. 1–5.

[313] A.N. Duc, D.S. Cruzes, G.K. Hanssen, T. Snarby, P. Abrahamsson: Coopetition of Software Firms in Open Source Software Ecosystems. In: *Software Business*, Springer, Cham, 2017 június 12. pp. 146–160.

[314] B. Vasilescu: Human Aspects, Gamification, and Social Media in Collaborative Software Engineering. In: *Companion Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA: ACM, 2014. pp. 646–649.

[315] B. Braunschweig, N. Dhage, M.J. Viera, C. Seaman, S. Sampath, G.A. Koru: Studying Volatility Predictors in Open Source Software. In: *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA: ACM, 2012. pp. 181–190.

[316] R. Subramanyam, M. Xia: Free/Libre Open Source Software development in developing and developed countries: A conceptual framework with an exploratory study. In: *Decision Support Systems*, 2008 december. Vol. 46, no. 1, pp. 173–186, ISSN 01679236. DOI 10.1016/j.dss.2008.06.006.

[317] C.-G. Wu, J.H. Gerlach, C.E. Young: An empirical analysis of open source software developers' motivations and continuance intentions. In: *Information & Management*, 2007 április. Vol. 44, no. 3, pp. 253–262, ISSN 03787206. DOI 10.1016/j.im.2006.12.006.

[318] H. Baytiyeh, J. Pfaffman: Open source software: A community of altruists. In: *Computers in Human Behavior*, 2010 november. Vol. 26, no. 6, pp. 1345–1354, ISSN 07475632. DOI 10.1016/j.chb.2010.04.008.

[319] E. Berdou: Insiders and Outsiders: Paid Contributors and the Dynamics of Cooperation in Community Led F/OS Projects. In: *Open Source Systems*, Springer, 2006., pp. 201–208. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/0-387-34226-5_20>

[320] W. Burleson, P. Tripathi: Mining Creativity Research to Inform Design Rationale in Open Source Communities. in: CARROLL, John M. (szerk.), In: *Creativity and Rationale*, London: Springer London, 2013., pp. 353–376. ISBN 978-1-4471-4110-5 978-1-4471-4111-2.

[321] B. Chawner: Community Matters Most: Factors That Affect Participant Satisfaction with Free/Libre and Open Source Software Projects. In: *Proceedings of the 2012 iConference*, New York, NY, USA: ACM, 2012. pp. 231–239.

[322] O. Nov, G. Kuk: Open source content contributors' response to free-riding: The effect of personality and context. In: *Computers in Human Behavior*, 2008 szeptember. Vol. 24, no. 6, pp. 2848–2861,

ISSN 07475632. DOI 10.1016/j.chb.2008.04.009.

[323] Y. Li, C.-H. Tan, H.-H. Teo, A.T. Mattar: Motivating Open Source Software Developers: Influence of Transformational and Transactional Leaderships. In: *Proceedings of the 2006 ACM SIGMIS CPR Conference on Computer Personnel Research: Forty Four Years of Computer Personnel Research: Achievements, Challenges & the Future*, New York, NY, USA: ACM, 2006. pp. 34–43.

[324] B. Vasilescu, A. Serebrenik, P. Devanbu, V. Filkov: How social Q&A sites are changing knowledge sharing in open source software communities., ACM Press, 2014. pp. 342–354.

[325] N. Viorres, P. Xenofon, M. Stavrakis, E. Vlachogiannis, P. Koutsabasis, J. Darzentas: Major HCI Challenges for Open Source Software Adoption and Development. In: *Online Communities and Social Computing*, Springer, 2007., pp. 455–464. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-73257-0_50>

[326] T. Tuunanen, J. Koskinen, T. Kärkkäinen: Automated software license analysis. In: *Automated Software Engineering*, 2009 december. Vol. 16, no. 3-4, pp. 455–490, ISSN 0928-8910, 1573-7535. DOI 10.1007/s10515-009-0054-z.

[327] I. Mistrík, J. Grundy, A. Van der Hoek, J. Whitehead: Collaborative Software Engineering: Challenges and Prospects. in: MISTRÍK, Ivan, GRUNDY, John, HOEK, André és WHITEHEAD, Jim (szerk.), In: *Collaborative Software Engineering*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010., pp. 389–403. ISBN 978-3-642-10293-6 978-3-642-10294-3.

[328] A. Bachmann, A. Bernstein: Software Process Data Quality and Characteristics: A Historical View on Open and Closed Source Projects. In: *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, New York, NY, USA: ACM, 2009. pp. 119–128.

[329] U. Raja: All complaints are not created equal: Text analysis of open source software defect reports. In: *Empirical Software Engineering*, 2013 február. Vol. 18, no. 1, pp. 117–138, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-012-9197-9.

[330] A.J. Ko, P.K. Chilana: How Power Users Help and Hinder Open Bug Reporting. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2010. pp. 1665–1674. [2015-

11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1753576>

[331] M. Gupta, A. Sureka: Nirikshan: Mining bug report history for discovering process maps, inefficiencies and inconsistencies., ACM Press, 2014. pp. 1–10.

[332] P. Anbalagan, M. Vouk: On Mining Data across Software Repositories. In: *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference On*, IEEE, 2009. pp. 171–174. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5069498>

[333] A. Tawileh, J. Hilton, S. Mcintosh: Modelling the Economics of Free and Open Source Software Security. In: *ISSE 2006—Securing Electronic Busines Processes*, Springer, 2006., pp. 326–335. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-8348-9195-2_35>

[334] V.B. Singh, K.K. Chaturvedi: Improving the Quality of Software by Quantifying the Code Change Metric and Predicting the Bugs. In: *Computational Science and Its Applications–ICCSA 2013*, Springer, 2013., pp. 408–426. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-39643-4_30>

[335] B. Rossi, B. Russo, G. Succi: Modelling Failures Occurrences of Open Source Software with Reliability Growth. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 268–280. [2014-05-31] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_21>

[336] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, C. Zhai: Have Things Changed Now?: An Empirical Study of Bug Characteristics in Modern Open Source Software. In: *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*, New York, NY, USA: ACM, 2006. pp. 25–33.

[337] S. Neuhaus, B. Plattner: Software Security Economics: Theory, in Practice. In: *The Economics of Information Security and Privacy*, Springer, 2013., pp. 75–92. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-39498-0_4>

[338] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, C. Zhai: Bug characteristics in open source software. In: *Empirical Software Engineering*, 2014 december. Vol. 19, no. 6, pp. 1665–1705, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-013-9258-8.

[339] P. Anbalagan, M. Vouk: On Reliability Analysis of Open Source Software - FEDORA., IEEE, 2008

november. pp. 325–326. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4700358>

[340] N. Iivari, H. Hedberg, T. Kirves: Usability in Company Open Source Software Context-Initial Findings from an Empirical Case Study. In: *Open Source Development, Communities and Quality*, Springer, 2008., pp. 359–365. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-0-387-09684-1_33>

[341] S. Datta, P. Sarkar, S. Das, S. Sreshtha, P. Lade, S. Majumder: How Many Eyeballs Does a Bug Need? An Empirical Validation of Linus' Law. In: *Agile Processes in Software Engineering and Extreme Programming*, Springer, 2014., pp. 242–250. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-319-06862-6_17>

[342] A. Ihara, M. Ohira, K.-i. Matsumoto: An Analysis Method for Improving a Bug Modification Process in Open Source Software Development. In: *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, ACM, 2009. pp. 135–144. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1595833>

[343] O. Arafat, D. Riehle: The Commenting Practice of Open Source. In: *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, ACM, 2009. pp. 857–864. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1640047>

[344] O. Gendreau, P.N. Robillard: A Process Practice to Validate the Quality of Reused Component Documentation: A Case Study Involving Open-Source Components. In: *Proceedings of the 2013 International Conference on Software and System Process*, ACM, 2013. pp. 61–69. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2486059>

[345] A. Oručević-Alagić, M. Höst: A Case Study on the Transformation from Proprietary to Open Source Software. In: *Open Source Software: New Horizons*, Springer, 2010., pp. 367–372. [2015-11-27] Retrieved: <http://link.springer.com/10.1007/978-3-642-13244-5_33>

[346] G. Schryen, R. Kadura: Open Source vs. Closed Source Software: Towards Measuring Security. In: *Proceedings of the 2009 ACM Symposium on Applied Computing*, ACM, 2009. pp. 2016–2023. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1529731>

[347] C.A. Ardagna, E. Damiani, F. Frati, S. Reale: Adopting Open Source for Mission-Critical Applications: A Case Study on Single Sign-On. In: *Open Source Systems*, Springer, 2006., pp. 209–220. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/0-387-34226-5_21>

[348] R.S. Kenett, X. Franch, A. Susi, N. Galanis: Adoption of Free Libre Open Source Software (FLOSS): A Risk Management Perspective., IEEE, 2014 július. pp. 171–180.

[349] A.-K. Groven, K. Haaland, R. Glott, A. Tannenberg: Security Measurements Within the Framework of Quality Assessment Models for Free/Libre Open Source Software. In: *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, New York, NY, USA: ACM, 2010. pp. 229–235.

[350] M. Galster, D. Tofan: Exploring Possibilities to Analyse Microblogs for Dependability Information in Variability-Intensive Open Source Software Systems. In: *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium On*, IEEE, 2013. pp. 321–325. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6688914>

[351] I. Samoladas, L. Angelis, I. Stamelos: Survival analysis on the duration of open source projects. In: *Information and Software Technology*, 2010 szeptember. Vol. 52, no. 9, pp. 902–922, ISSN 09505849. DOI 10.1016/j.infsof.2010.05.001.

[352] M.J. Scialdone, N. Li, R. Heckman, K. Crowston: Group Maintenance Behaviors of Core and Peripherial Members of Free/Libre Open Source Software Teams. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 298–309. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_26>

[353] M. Foulonneau, R. Pawelzik, B. Grégoire, O. Donak: Analyzing the Open Source Communities' Lifecycle with Communication Data. In: *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*, ACM, 2013. pp. 340–344. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2536183>

[354] D. Di Ruscio, D.S. Kolovos, I. Korkontzelos, N. Matragkas, J.J. Vinju: OSSMETER: A software measurement platform for automatically analysing open source software projects., ACM Press, 2015. pp. 970–973.

[355] L. Lavazza, S. Morasca, D. Taibi, D. Tosi: An Empirical Investigation of Perceived Reliability of Open Source Java Programs. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ACM, 2012. pp. 1109–1114. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2231951>

[356] Q.C. Taylor, J.L. Krein, A.C. MacLean, C.D. Knutson: An Analysis of Author Contribution Patterns in Eclipse Foundation Project Source Code. In: *Open Source Systems: Grounding Research*, Springer, 2011., pp. 269–281. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-24418-6_19>

[357] S. Counsell, S. Swift: An Empirical Study of Potential Vulnerability Faults in Java Open-Source Software. In: *Innovative Techniques in Instruction Technology, E-Learning, E-Assessment, and Education*, Springer, 2008., pp. 514–519. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-1-4020-8739-4_91>

[358] S. Bajracharya, J. Ossher, C. Lopes: Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. In: *Science of Computer Programming*, 2014 január. Vol. 79, pp. 241–259, ISSN 01676423. DOI 10.1016/j.scico.2012.04.008.

[359] R.K. McLean: Comparing Static Security Analysis Tools Using Open Source Software., IEEE, 2012 június. pp. 68–74.

[360] G.M. Kapitsaki, N.D. Tselikas, I.E. Foukarakis: An insight into license tools for open source software systems. In: *Journal of Systems and Software*, 2015 április. Vol. 102, pp. 72–87, ISSN 01641212. DOI 10.1016/j.jss.2014.12.050.

[361] M. Tiemann: An objective definition of open standards. In: *Computer Standards & Interfaces*, 2006 június. Vol. 28, no. 5, pp. 495–507, ISSN 09205489. DOI 10.1016/j.csi.2004.12.003.

[362] S. Crane, S. Pearson: Security Trustworthiness Assessment of Platforms. In: *Digital Privacy*, Springer, 2011., pp. 457–483. [2015-11-27] Retrieved: <http://link.springer.com/content/pdf/10.1007/978-3-642-19050-6_17.pdf>

[363] A. Ribeiro, P. Meirelles, N. Lago, F. Kon: Ranking Source Code Static Analysis Warnings for Continuous Monitoring of FLOSS Repositories. In: *Open Source Systems: Enterprise Software and Solutions*, Springer, Cham, 2018 június 8. pp. 90–101.

[364] G. Robles, J.M. González-Barahona, C. Cervigón, A. Capiluppi, D. Izquierdo-Cortázar: Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 2014. pp. 222–231. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=2597107>

[365] M. Bruntink: Towards base rates in software analytics. In: *Science of Computer Programming*, 2015 január. Vol. 97, pp. 135–142, ISSN 01676423. DOI 10.1016/j.scico.2013.11.023.

[366] M. Conklin, J. Howison, K. Crowston: Collaboration Using OSSmole: A Repository of FLOSS Data and Analyses. In: *ACM SIGSOFT Software Engineering Notes*, ACM, 2005. pp. 1–5. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1083164>

[367] L. Lavazza, S. Morasca, D. Taibi, D. Tosi: Predicting OSS Trustworthiness on the Basis of Elementary Code Assessment. In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 2010. pp. 36. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1852834>

[368] M. Yang, J. Wu, S. Ji, T. Luo, Y. Wu: Pre-Patch: Find Hidden Threats in Open Software Based on Machine Learning Method. In: *Services – SERVICES 2018*, Springer, Cham, 2018 június 25. pp. 48–65.

[369] A. Stump: *Verified Functional Programming in Agda*, New York, NY, USA: Association for Computing Machinery and Morgan & Claypool, 2016. ISBN 978-1-970001-27-3.

[370] A. Ruef, C. Rohlf: Programming Language Theoretic Security in the Real World: A Mirage or the Future? In: *Cyber Warfare*, Springer, Cham, 2015., Advances in information security. pp. 307–321. ISBN 978-3-319-14038-4 978-3-319-14039-1.

[371] D. Terei, S. Marlow, S. Peyton Jones, D. Mazières: Safe Haskell. In: *Proceedings of the 2012 Symposium on Haskell Symposium*, 2012. pp. 137–148. [2014-01-12] Retrieved: <http://dl.acm.org/citation.cfm?id=2364524>

[372] R. Kemp: Open source software (OSS) governance in the organisation. In: *Computer Law & Security Review*, 2010 május. Vol. 26, no. 3, pp. 309–316, ISSN 02673649. DOI 10.1016/j.clsr.2010.01.008.

[373] S.E. Sim, E.B. Stenberg: Intellectual Property Law in Source Code Reuse and Remix. In: *Finding*

Source Code on the Web for Remix and Reuse, Springer, 2013., pp. 311–322. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-1-4614-6596-6_16>

[374] D. Bahn, D. Dressel: Liability and Control Risks with Open Source Software. In: *Information Technology: Research and Education, 2006. ITRE'06. International Conference On*, IEEE, 2006. pp. 242–245. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4266334>

[375] D.M. German, J.M. González-Barahona: An Empirical Study of the Reuse of Software Licensed under the GNU General Public License. In: *Open Source Ecosystems: Diverse Communities Interacting*, Springer, 2009., pp. 185–198. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-02032-2_17>

[376] W. Scacchi, T.A. Alspaugh: Understanding the role of licenses and evolution in open architecture software ecosystems. In: *Journal of Systems and Software*, 2012 július. Vol. 85, no. 7, pp. 1479–1494, ISSN 01641212. DOI 10.1016/j.jss.2012.03.033.

[377] G.R. Gangadharan, M. Butler: Free and Open Source Software Adoption in Emerging Markets: An Empirical Study in the Education Sector. In: *Open Source Systems: Long-Term Sustainability*, Springer, 2012., pp. 244–249. ISBN 978-3-642-33442-9.

[378] M. Caneill, D.M. Germán, S. Zacchiroli: The Debsources Dataset: Two decades of free and open source software. In: *Empirical Software Engineering*, 2017 június 1. Vol. 22, no. 3, pp. 1405–1437, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-016-9461-5.

[379] Y. Wu, Y. Manabe, D.M. German, K. Inoue: How are Developers Treating License Inconsistency Issues? A Case Study on License Inconsistency Evolution in FOSS Projects. In: *Open Source Systems: Towards Robust Practices*, Springer, Cham, 2017 május 22. pp. 69–79.

[380] Y. Wu, Y. Manabe, T. Kanda, D.M. German, K. Inoue: Analysis of license inconsistency in large collections of open source projects. In: *Empirical Software Engineering*, 2017 június 1. Vol. 22, no. 3, pp. 1194–1222, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-016-9487-8.

[381] M. Schuenck, Y. Negócio, G. Elias, S. Miranda, J. Dias Jr, G. Cavalcanti: X-ARM: A Step towards Reuse of Commercial and Open Source Components. In: *Reuse of Off-the-Shelf Components*, Springer, 2006., pp. 432–435. [2015-11-26] Retrieved: <http://link.springer.com/chapter/10.1007/11763864_40>

[382] G.M. Kapitsaki, F. Kramer: Open Source License Violation Check for SPDX Files. In: *Software Reuse for Dynamic Systems in the Cloud and Beyond*, Springer, 2014., pp. 90–105. ISBN 978-3-319-14130-5.

[383] C. Vendome, G. Bavota, M.D. Penta, M. Linares-Vásquez, D. German, D. Poshyvanyk: License usage and changes: A large-scale study on gitHub. In: *Empirical Software Engineering*, 2017 június 1. Vol. 22, no. 3, pp. 1537–1577, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-016-9438-4.

[384] G.R. Gangadharan, V. D'Andrea, S. Paoli, M. Weiss: Managing license compliance in free and open source software development. In: *Information Systems Frontiers*, 2012 április. Vol. 14, no. 2, pp. 143–154, ISSN 1387-3326, 1572-9419. DOI 10.1007/s10796-009-9180-1.

[385] T. Tuunanen, J. Koskinen, T. Kärkkäinen: Retrieving Open Source Software Licenses. In: *Open Source Systems*, Springer, 2006., pp. 35–46. ISBN 978-0-387-34226-9.

[386] D.M. German, Y. Manabe, K. Inoue: A Sentence-Matching Method for Automatic License Identification of Source Code Files. In: *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ACM, 2010. pp. 437–446. [2015-11-24] Retrieved: <http://dl.acm.org/citation.cfm?id=1859088>

[387] D.A. Almeida, G.C. Murphy, G. Wilson, M. Hoye: Investigating whether and how software developers understand open source software licensing. In: *Empirical Software Engineering*, 2018 április 27. pp. 1–29, ISSN 1382-3256, 1573-7616. DOI 10.1007/s10664-018-9614-9.

[388] C. Santos, G. Kuk, F. Kon, J. Pearson: The attraction of contributors in free and open source software projects. In: *The Journal of Strategic Information Systems*, 2013 március. Vol. 22, no. 1, pp. 26–45, ISSN 09638687. DOI 10.1016/j.jsis.2012.07.004.

[389] R. Sen, S.S. Singh, S. Borle: Open source software success: Measures and analysis. In: *Decision Support Systems*, 2012 január. Vol. 52, no. 2, pp. 364–372, ISSN 01679236. DOI 10.1016/j.dss.2011.09.003.

[390] DAMIANI, Ernesto, FRATI, Fulvio, RIEHLE, Dirk és WASSERMAN, Anthony I. (szerk.): *Open Source Systems: Adoption and Impact*, Cham: Springer International Publishing, 2015. IFIP Advances in Information and Communication Technology. ISBN 978-3-319-17836-3 978-3-319-17837-0.

[391] P.T. Breuer, S. Pickin: Open source verification in an anonymous volunteer network. In: *Science*

of Computer Programming, 2014 október. Vol. 91, pp. 161–187, ISSN 01676423. DOI 10.1016/j.scico.2013.08.010.

[392] C. Amega-Selorm, J. Awotwi: Free and Open Source Software (FOSS): It's Significance or Otherwise to the e-Governance Process in Ghana. In: *Proceedings of the 4th International Conference on Theory and Practice of Electronic Governance*, New York, NY, USA: ACM, 2010. pp. 91–95.

[393] B. Rossi, B. Russo, G. Succi: Download Patterns and Releases in Open Source Software Projects: A Perfect Symbiosis? In: *Open Source Software: New Horizons*, Springer, 2010., pp. 252–267. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-642-13244-5_20>

[394] G. Schryen: A Comprehensive and Comparative Analysis of the Patching Behavior of Open Source and Closed Source Software Vendors., IEEE, 2009. pp. 153–168.

[395] L. Yu, K. Chen: Evaluating the Post-Delivery Fault Reporting and Correction Process in Closed-Source and Open-Source Software. In: *Software Quality, 2007. WoSQ'07: ICSE Workshops 2007. Fifth International Workshop On*, IEEE, 2007. pp. 8–8. [2015-11-24] Retrieved: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4273475>

[396] LASSENIUS, Casper, DINGSØYR, Torgeir és PAASIVAARA, Maria (szerk.): *Agile Processes, in Software Engineering, and Extreme Programming*, Cham: Springer International Publishing, 2015. Lecture Notes in Business Information Processing. ISBN 978-3-319-18611-5 978-3-319-18612-2.

[397] A. Christl: Free Software and Open Source Business Models. In: *Open Source Approaches in Spatial Data Handling*, Springer, 2008., pp. 21–48. [2015-11-27] Retrieved: <http://link.springer.com/chapter/10.1007/978-3-540-74831-1_2>

[398] Mike Wheatley: Microsoft opens first Transparency Center in Europe. In: *SiliconANGLE*. 2015 június 8. [2018-08-14] Retrieved: <https://siliconangle.com/2015/06/08/microsoft-opens-first-transparency-center-in-europe/>

[399] R. Méndez-Durón: Do the allocation and quality of intellectual assets affect the reputation of open source software projects? In: *Information & Management*, 2013 november. Vol. 50, no. 7, pp. 357–368, ISSN 03787206. DOI 10.1016/j.im.2013.05.006.

[400] Reproducible-Builds.Org.. [2018-08-14] Retrieved: <https://reproducible-builds.org/>

[401] ReproducibleBuilds. In: *Debian Wiki*. [2018-09-17] Retrieved: <https://wiki.debian.org/ReproducibleBuilds>

[402] F. Grobert, A.-R. Sadeghi, M. Winandy: Software Distribution as a Malware Infection Vector. In: *Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference For*, IEEE, 2009. pp. 1–6.

[403] A. Zerouali, E. Constantinou, T. Mens, G. Robles, J. González-Barahona: An Empirical Analysis of Technical Lag in npm Package Dependencies. In: *New Opportunities for Software Reuse*, Springer, Cham, 2018 május 21. pp. 95–110.

[404] A. Ojamaa, K. Düüna: Security Assessment of Node.Js Platform. In: *Information Systems Security*, Springer, Berlin, Heidelberg, 2012 december 15. pp. 35–43.

[405] C.-S. Kuo, C.-Y. Huang, S.-P. Luan: A Study of Using Two-Parameter Generalized Pareto Model to Analyze the Fault Distribution of Open Source Software., IEEE, 2012 június. pp. 88–97.

[406] J. Wu, C. Wang, X.-x. Jia, C. Liu: Mining Open Source Component Behavior for Reuse Evaluation. In: *High Confidence Software Reuse in Large Systems*, Springer, 2008., pp. 112–115. ISBN 978-3-540-68073-4.

[407] A. Capiluppi, J.M. González-Barahona, I. Herraiz, G. Robles: Adapting the ”Staged Model for Software Evolution” to Free/Libre/Open Source Software. In: *Ninth International Workshop on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting*, New York, NY, USA: ACM, 2007. pp. 79–82.

[408] L. Lavazza, S. Morasca, D. Taibi, D. Tosi: OP2A: How to Improve the Quality of the Web Portal of Open Source Software Products. In: *Web Information Systems and Technologies*, Springer, 2012., pp. 149–162. ISBN 978-3-642-28082-5.

[409] Y. Roumani, J.K. Nwankpa, Y.F. Roumani: Adopters' Trust in Enterprise Open Source Vendors: An Empirical Examination. In: *Journal of Systems and Software*, 2017 március 1. Vol. 125, pp. 256–270, ISSN 0164-1212. DOI 10.1016/j.jss.2016.12.006.

[410] L. Aversano, M. Tortorella: Quality evaluation of floss projects: Application to ERP systems. In:

Information and Software Technology, 2013 július. Vol. 55, no. 7, pp. 1260–1276, ISSN 09505849. DOI 10.1016/j.infsof.2013.01.007.

[411] L. López, D. Costal, C.P. Ayala, X. Franch, M.C. Annosi, R. Glott, K. Haaland: Adoption of OSS components: A goal-oriented approach. In: *Data & Knowledge Engineering*, 2015 szeptember. Vol. 99, pp. 17–38, ISSN 0169023X. DOI 10.1016/j.datak.2015.06.007.

[412] K. Gerberding: NIST, CIS/SANS 20, ISO 27001: What's the difference? In: *Hitachi Systems Security | Managed Security Services Provider*. 2017 szeptember 5. [2018-09-17] Retrieved: <https://www.hitachi-systems-security.com/blog/nist-cissans-20-iso-27001-simplifying-security-control-assessments/>

[413] National Institute of Standards and Technology: NIST Cybersecurity White Paper: *Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1*. Gaithersburg, MD. National Institute of Standards and Technology, 2018. [2018-08-20] Retrieved: <http://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf>

Rövidítések jegyzéke

ABI Application Programming Interface, a szoftver vagy szoftverrendszer egyes részei között definiált interfész vagy kommunikációs protokoll, melynek elsődleges célja a szoftver karbantartásának és implementációjának leegyszerűsítése.

ABI Application Binary Interface, két program modul között értelmezett bináris interfész. Az egyik fél gyakran az operációs rendszer valamely szolgáltatása.

CC Common Criteria, információbiztonsági tanúsításra használható elterjedt keretrendszer, ahol a rendszer használói meghatározhatják a teljesítendő funkcionális és minősítési követelményeket, a gyártó megvalósíthatja, végül egy független minősítő laboratórium ellenőrizheti azokat.

COBIT Control Objectives for Information and Related Technology egy, az ISACA (Information Systems Audit and Control Association) által definiált IT keretrendszer.

COTS Commercial Off-the-Shelf. Olyan IT termék, amelyet módosítás, testreszabás nélkül árusítanak és használnak fel. A COTS termékeket könnyen telepíthetőre tervezik és általában egyszerűen illeszthetők a már meglévő termékekhez. (Az átlagos számítógép felhasználó által használt szinte valamennyi program ebbe a kategóriába esik). A COTS szoftverek előnye az tömegtermelésből adódó alacsony ár.

DDOS A Distributed Denial-of-Service avagy szolgáltatás megtagadással járó támadás célja a kiszemelt kiszolgáló, hálózat vagy szolgáltatás normál adatforgalmának ellehetetlenítése nagy mennyiségű Internetes forgalom generálásával.

LIRE Létfontosságú Információs Rendszerelemek

LRE Létfontosságú Rendszerelemek

FLOSS Free Libre and Open Source Software, a Szabad Szoftver nemzetközileg ismert betűszava. A nyílt fejlesztési modell terméke, amely meghatározott feltételeket kielég.

FLOSS elem A bővített FLOSS kategóriába tartozó bármilyen programkód, abból képzett bináris, konfigurációs elem vagy metainformáció. Ide tartozik a szabad szoftver, és szervezet vagy beszállítói által integrált nyílt forráskódú komponens is.

GPG GNU Privacy Guard, a PGP (Pretty Good Privacy) kriptográfiai szoftvercsomag gyakran használt RFC 4880 kompatibilis FLOSS alternatívája.

GPL GNU General Public License. Széles körben elterjedt kötött (restrictive) FLOSS licencforma, amely a végfelhasználónak lehetővé teszi a termék szabad futtatását, tanulmányozását, megosztását és módosítását.

OSSD Open Source Software Development, a nyílt fejlesztési modell egyedi jellemzőkkel rendelkező szoftveralkotási folyamata, melynek alapja a fejlesztőközösség, terméke a Szabad Szoftver

IoT Internet of Things, a dolgok internete. Egymással kommunikációs kapcsolatban álló hardver és szoftver elemek együttese, amelyeket hétkoznapi dolgok "intelligensé" tételére használunk fel. Jellemző példái az "okos-eszközök", viselhető elktronika (wareware), intelligens orvosi szenzorok és segédeszközök, épületautomatizálás, megoszló közlekedés irányítás.

LIRE Létfontosságú Információs Rendszerelem és létesítmény a társadalom olyan hálózatszerű, fizikai vagy virtuális rendszerei, eszközei és módszerei, amelyek az információ folyamatos biztosítása és az informatikai feltételek üzemfolytonosságának szükségességéből adódóan önmagukban létfontosságú rendszerelemek, vagy más azonosított létfontosságú rendszerelemek működéséhez nélkülözhetetlenek.

NIST National Institute of Standards and Technology, az Egyesült Államok szabványokkal karbantartásával és terjesztésével foglalkozó szervezete.

OSSD Open Source Software Development, azaz nyílt forráskódú szoftverfejlesztés. Ebben az értekezésben mint egyedi fejlesztési módszertan értelmezendő.

SCRM Supply Chain Risk Management, beszállítói lánccal kapcsolatos kockázatelemzési módszerek gyűjtőneve.

SEO Search Engine Optimalization, weboldalak és azok kapcsolatainak optimalizálása a keresőtalálatban elért helyezések optimalizálása céljából. Egyes változatai (Black SEO) a kereső gyártója által tiltott vagy ellenjavalt módszereket is alkalmaznak.

SRGM Software Reliability Growth Model, szoftverek megbízhatóságának és fenntarthatóságának vizs-

gálatát célzó modellek.

WoT Web of Trust. A PKI alapú hitelesség ellenőrzés alternatív formája, ahol szigorú hierarchia és aláírás láncolat alkalmazása helyett gráfszerű hálózatot és többszörös aláírást alkalmaznak. A megbízhatóságot nem a gyökértanúsítóba vetett bizalom, hanem a megbízhatónak minősített aláírókhoz vezető útak száma biztosítja.

Ábrák jegyzéke

Táblázatok jegyzéke

Függelék

8.1. táblázat: Értekezés előállításához felhasznált nyílt forrású technológiák

Szoftver	kutatási folyamat
Inkscape	ábrák, pdf manipuláció
Vue	gondolattérképek, folyamatábrák
Zotero	publikációkezelés
SQLite	adatbázisok
PHP	publikáció kategorizáló keretrendszer
Python	adatmigráció
JavaScript	vizualizáció
LaTeX	szedés és kiadványszerkesztés
pandoc	markdown konverzió
Gentoo Linux	operációs rendszer