ferenc.rudolf_87_23

Továbbfejlesztett szoftverhiba előrejelzés fogalmi metrikák és gépi tanulás segítségével

Improved Bug Prediction Through
Conceptual Metrics and Machine Learning

MTA Doktora értekezés tézisei

Ferenc Rudolf

Szeged, 2023

ferenc.rudolf_87_23

Tartalomjegyzék

Az értekezés célkitűzései

A szoftverek minőségének hosszú története tele van sokkoló pillanatokkal [17, 25, 27, 39], amelyek után nem ritkák az olyan vallomások, mint a következő: „Ha jobban odafigyeltünk volna a minőségre, ez soha nem történhetett volna meg. . . ”. Szerencsére a szoftvercégek egyre inkább felismerik a minőségbiztosítás fontosságát, és manapság már különböző monitoring technikákat alkalmaznak, hogy naprakész képet kapjanak a rendszereik minőségéről. Ez a felismerés a kutatókat is a különböző szoftverminőségi jellemzők közötti összetett összefüggések feltárása és megértése felé terelte.

A szoftverminőséggel kapcsolatos két legalapvetőbb, mégis legnehezebb kérdés: 1) mi is az pontosan, és 2) hogyan tudjuk mérni. E kérdések megválaszolásához a minőséget kisebb összetevőkre kell bontanunk, amelyek a különböző aspektusait írják le. Különböző szabványok - mint például az ISO/IEC 9126 vagy annak utódja, az ISO/IEC 25010 - születtek e kihívások megoldására. Ezek a szabványok azonban csak egy magasabb absztrakciós szint szempontjait javasolják (pl. karbantarthatóság és megbízhatóság), anélkül, hogy pontosan meghatároznák az alacsony szintű attribútumokat. Következésképpen számos megvalósítást mutattak be e probléma megoldására, beleértve a szoftverminőségi modelleket [1]. A minőségi modellek pontos forráskód mérőszámokra támaszkodnak a rendszer különböző aspektusainak, például a méretnek, a komplexitásnak, a dokumentációnak, a csatolásnak és a kohéziónak a mérésére. Ezek az alacsony szintű attribútumok közvetlenül a forráskódból mérhetők, és a magasabb szintekre is átvihetők a minőség olyan részjellemzőinek mérése érdekében, mint a stabilitás és a tesztelhetőség (amelyek viszont a karbantarthatóság - a minőség fontos összetevője - részjellemzői).

Az egyik legkorábbi felismerés az, hogy a forráskód mérőszámok (metrikák) a minőség leírásának kulcsfontosságú összetevői. A kohézió és csatolás metrikák bizonyították hasznosságukat különböző területeken, mint például a hibaelőrejelzés [5] és a hatáselemzés [12], amelyek közvetlen hatással vannak a szoftver stabilitására és tesztelhetőségére. A forráskód szerkezetét megragadó hagyományos csatolás és kohézió metrikák mellett fogalmi megfelelőiket is meghatároztuk a rendszer absztraktabb mérésének megragadására. Ebben az értekezésben bemutatjuk az általunk definiált fogalmi kohézió és csatolás metrikákat, és igazoljuk azok hasznosságát a hibák előrejelzésében és a hatáselemzésben.

A szoftverek másik minőségi mutatója a hibák száma. A meglévő szoftverek karbantartásának költségei gyakran meghaladják a kezdeti fejlesztés költségeit [20]. Ezért a szoftverhibák mielőbbi megtalálása rendkívül fontos, mert még ha minden bevált jó gyakorlatot be is tartunk, naprakész komponenseket használunk és a legújabb programozási nyelv verziót alkalmazzuk, akkor is emberek vagyunk, így még a legszigorúbb felülvizsgálati folyamat mellett is bekerülhet egy hiba a szoftverbe. E hibák mind a szoftverfejlesztő cégeknek, mind a szoftverfelhasználóknak (pénzügyi és hírnévbeli) károkat okozhatnak. A szoftverhibák megtalálása rendkívül időigényes lehet, még akkor is, ha pontosan erre a célra tervezett statikus vagy dinamikus elemző eszközöket használunk. Ezek az eszközök gyakran nem veszik észre a valódi hibákat, és magas a hamis találatok aránya is. Ezért a fejlesztők gyakran nem veszik komolyan az eszközök figyelmeztetéseit.

Manapság mesterséges intelligencia alapú megoldások is alkalmazhatók a szoftverhibák előrejelzésére. A mesterséges intelligencia algoritmusok hatékony betanításához azonban jelentős mennyiségű adatra van szükség. Szerencsére a nyílt forráskódú szoftverek (és az olyan verziókezelő rendszerek, mint a GitHub vagy a GitLab) elterjedésével a kutatók a szoftverfejlesztés történeti adatainak gazdag halmazához férnek hozzá. Az információk azonban különböző formátumokban és rendszerekben állnak rendelkezésre (még ugyanazon a projekten belül is), és az összegyűjtésük nehézkes. Emiatt léteznek olyan hibaadatbázisok, amelyek különböző rendszerekből származó

szoftverhibák gyűjteményét tartalmazzák. Ezek azonban különböző formátumokban és tartalommal állnak rendelkezésre, ami kihívást jelent a mesterséges intelligencia területén történő együttes felhasználásukban. Ebben a munkában bemutatjuk, hogyan kombináltuk és bővítettük a meglévő hibaadatbázisokat a gépi tanulási algoritmusok futtatásához, hogy a lehető legjobb eredményt kapjuk.

Tudjuk, hogy a mesterséges intelligencia esetében minél több adatunk van, annál jobb. Az egyik legkiemelkedőbb módszer az egyre népszerűbbé váló mély mesterséges neurális hálózatok, amelyek hatalmas adatmennyiséggel taníthatók be sikeresen. Fontos eredmény, amit ebben a értekezésben bemutatunk, az általunk létrehozott egységes hibaadatbázis, amit több hagyományos gépi tanulási algoritmus kiértékelésére használtunk. Azt is részletesen bemutatjuk, hogy a forráskód metrikák hogyan használhatók mély neurális hálózatok segítségével hibák előrejelzésére. Lépésről lépésre ismertetjük a folyamatot, beleértve a jó eredmény eléréséhez szükséges összes finomhangolást.

Az értekezés tézispontjait fejezetenként mutatom be a füzetben. Összefoglalva:

T1 **Fogalmi kohézió hiba előrejelzéshez.** Ismertetem a $C3$ fogalmi kohézió metrikát, majd főkomponens analízis segítségével megmutatom, hogy önálló dimenziót alkot a strukturális kohézió metrikákhoz képest. Továbbá megmutatom, hogy a $C3$ metrika javítja a strukturális kohézió metrikák hiba előrejelző képességeit [6].

T2 **Fogalmi csatolás hatásanalízishez.** Ismertetem a $CCBC$ és $CCBC_m$, valamint a $CoCC$ és $CoCC_m$ fogalmi csatolás metrikákat, majd megmutatom, hogy hatásanalíziskor a $CCBC_m$ metrika rangsorolja a legjobban a kapcsolódó osztályokat messze megelőzve a strukturális csatolás metrikákat ebből a szempontból [7].

T3 **Új fogalmi csatolás és kohézió metrikák.** Definiálom a paraméterezhető $CCBO$ és $CLOM5$ új fogalmi metrikákat, melyeket könnyebb kiszámolni, mint a strukturális megfelelőiket, miközben a hiba előrejelző képességeik nagyon hasonlóak. Megmutatom továbbá azt is, hogy e fogalmi metrikák javítják a strukturális metrikák hiba előrejelző képességeit [9].

T4 **Egységesített publikus hibaadatbázis hibák előrejelzéséhez.** Összefoglalom, hogyan kutattuk fel az összes elérhető, széles körben használt Java hibaadatbázist, összehasonlítva és kombinálva őket, hogy létrehozzunk egy egységes hibaadatbázist, amelyet számos általunk kiszámított forráskód metrikával egészítettünk ki. Megmutatom, hogy az egységesített hibaadatbázis jól használható hiba előrejelző gépi tanuló modellek készítéséhez (projektenként, összesítve és projektek között) és kiértékelem az eredményeket. [4].

T5 **Mélytanulás a hiba előrejelzéshez.** Részletes módszertant adok a mély neurális hálók hiba előrejelzésben történő alkalmazására és optimális hiperparamétereinek megtalálására. Az egységesített hibaadatbázison mély neuronhálót építek és összehasonlítom a hagyományos gépi tanuló algoritmusok eredményeivel. Megmutatom, hogy csak a véletlen erdő tudja felülmúlni. E kettő kombinációja adja végül a legjobb eredményt. Megmutatjuk azt is, hogy több tanuló adattal várhatóan tovább javul a mély neuronháló teljesítménye [2].

1. Fogalmi kohézió hiba előrejelzéshez

A szoftverek modularizálása, különösen az objektum orientált szemlélet a forráskód szervezésének és érthetőségének javítására szolgáló megközelítés. A szoftverfejlesztőknek a rendszert alkotó osztályok jól összefüggő reprezentációját kell létrehozniuk, amelyben mind az osztályok szerepe, mind a közöttük lévő kapcsolatok tisztázása alapvető fontosságú. Az objektum orientált elemzés és tervezés egyik fontos célja olyan rendszer létrehozása, amelyben az osztályok nagyfokú kohézióval rendelkeznek, és alacsony a köztük lévő csatolás. A szoftver ilyen módon történő felépítése egyebek között jelentősen egyszerűsíti a megértést, a tesztelést, az újrafelhasználhatóságot, és a karbantarthatóságot.

Egy szoftvermodul kohéziója úgy definiálható, mint annak mértéke, hogy a modul elemei mennyire tartoznak össze [10]. A kohézió fogalmi szempontból is vizsgálható, amely azt fejezi ki, hogy az adott osztály milyen szinten reprezentálja a követelmény vagy a specifikációs dokumentációkban leírtakat. Az objektum orientált szoftverekben a kohéziót általában osztályszinten mérik, és számos egymástól eltérő kohézió metrikát javasoltak [10, 11, 14, 34], amelyek a kohézió különböző aspektusait próbálják megragadni, vagy amelyek a kohézió egy adott értelmezését tükrözik. A kohéziót általában a forráskódból kinyert strukturális információkkal mérik, amelyek azt a mértéket rögzítik, hogy egy osztály elemei strukturális szempontból mennyire tartoznak össze. A strukturális metrikák ugyanakkor nem adnak támpontot arra vonatkozólag, hogy az osztály elemei fogalmilag mennyire összefüggőek.

Az osztályok kohéziójának mérésére egy új metrikát javasoltunk, amelyet az osztályok fogalmi kohéziójának (Conceptual Cohesion of Classes, röviden $C3$) neveztünk el. Ez a metrika azt méri, hogy egy osztály metódusai fogalmi szempontból milyen erősen kapcsolódnak egymáshoz. A metódusok közötti fogalmi kapcsolat a szöveges koherencia elvére épül [30]. Megközelítésünk azon az előfeltevésen alapul, hogy a forráskódba ágyazott strukturálatlan információk (megjegyzések, azonosító nevek, stb.) tükrözik a szoftver probléma és alkalmazási terület fogalmi viszonyait, valamint a program számítási logikáját. Ezek az információk a szoftver alkalmazási területének szemantikáját tükrözik, és a programozási nyelv alap szemantikáján túl egy új szemantikai réteget adnak hozzá a forráskódhoz.

A forráskódból származó strukturálatlan információk kinyeréséhez és elemzéséhez az LSI (Latent Semantic Indexing) [16], egy fejlett információ kinyerési módszert használtunk. A természetes nyelvekben egy szöveg koherenciájának mérésére az LSI-t a szövegben egymás után következő mondatok közötti hasonlósági mértékek kiszámítására használják. Két egymást követő mondat közötti nagy hasonlóság azt jelzi, hogy a két mondat összefügg, míg az alacsony hasonlóság a téma megszakadását jelzi. A teljes szöveg koherenciájának mérésére az egymást követő mondatokból származó hasonlósági mértékek átlagát szokás használni.

Az LSI szoftverek forráskódjának vizsgálatához történő adaptációjához az egyes természetes nyelvekben használt fogalmak megfelelőit kell meghatározni a forráskódra vonatkozólag. A természetes nyelvek esetében az indexelendő szövegegységek (dokumentumok) a mondatok, a bekezdések, illetőleg a szakaszok. Az forráskódban a metódusokat tekintjük dokumentumoknak [32, 33, 36], amelyet az LSI módszer alapján indexelünk. Az egyes metódusokban számunkra érdekes információt a megjegyzések és az egyes komponensek elnevezése hordoz [31]. A hasonlóság kiszámításához minden, az adott osztályban lévő metóduspárt felhasználunk, ezzel m_k és m_j közötti fogalmi hasonlóságot kapjuk $CSM(m_k, m_j)$ minden $m_k, m_j \in M$-re, ahol M az adott osztály metódusainak halmaza. Ezt a módosítást az eredeti módszerhez képest, amelyben az egymást követő dokumentumok hasonlóságát számítjuk az indokolja, hogy egy adott osztály az objektum orientált tervezés alapelvei szerint egy adott koncepciót valósít meg, egy helyesen

felépített osztály metódusai koherensek. Az így kapott hasonlósági mértékeket átlagoljuk az osztály metódusainak számával. A $C3$ metrika az így kapott átlagolt fogalmi hasonlóság mértékével egyezik, amennyiben az pozitív, egyébként 0-nak vesszük.

Az új metrika alkalmazhatóságát két esettanulmányon keresztül vizsgáltuk. Az első esettanulmány célja az volt, hogy megállapítsuk, hogy a $C3$ vajon ad-e hozzá lényegi új információt a kohézió méréséhez a létező strukturális kohézió metrikákhoz képest. A második esettanulmányban a $C3$-mat összehasonlítottuk a korábbi strukturális kohézió metrikákkal, továbbá a $C3$ és a strukturális kohézió metrikák kombinációit a pusztán strukturális metrikák kombinációival az osztályok hibáinak előrejelzésében. Azt vizsgáltuk, hogy melyik metrikahalmaz a leghatékonyabb prediktor.

Az első esettanulmányban főkomponens analízist (PCA) hajtottunk végre a Mozilla rendszer forráskódján számolt különböző kohézió metrikákra nézve. A PCA hat főkomponenst (PC) mutatott ki, amelyek az adatkészlet varianciájának 96%-át írják le. Az egyes főkomponensek hozzájárulását az 1. táblázat tartalmazza. A PCA eredmények azt mutatják, hogy a $C3$ egy saját dimenziót határoz meg; a $C3$ az egyetlen jelentős faktor a PC4-ben. Ezek az eredmények statisztikailag alátámasztják hipotézisünket, miszerint a $C3$ kohézió mérték az esettanulmányban mért összes metrika által meghatározott osztály kohézió mértékének eltérő aspektusait ragadja meg.

1. táblázat. Főkomponens együtthatók.

	PC_1	**PC_2**	**PC_3**	**PC_4**	**PC_5**	**PC_6**
Proportion	29.6	20.91	10.12	10.04	17.0	8.56
Cumulative	29.6	50.51	60.63	70.67	87.67	96.24
C3	-0.061	-0.037	-0.017	**0.996**	-0.043	0.008
LCOM1	**0.922**	-0.001	0.052	-0.032	0.317	-0.012
LCOM2	**0.914**	-0.018	0.044	-0.029	0.331	0.004
LCOM3	**0.609**	-0.129	0.052	-0.048	**0.736**	-0.138
LCOM4	0.206	-0.196	-0.001	-0.036	**0.937**	-0.102
LCOM5	0.084	0.032	**0.995**	-0.017	0.018	-0.040
ICH	**0.914**	0.056	0.066	-0.057	-0.065	-0.144
TCC	-0.023	**0.933**	-0.033	-0.002	-0.116	0.283
LCC	0.045	**0.966**	0.079	-0.050	-0.136	0.095
Coh	-0.118	0.476	-0.061	0.012	-0.176	**0.846**

A második esettanulmányban annak elemzésére összpontosítunk, hogy a kohézió mérőszámok milyen mértékben használhatók a hibák előrejelzésére. Adataink elemzéséhez egyváltozós és többváltozós logisztikus regressziós elemzési módszereket választottunk. Az egyváltozós módszerrel az egyes metrikák hatását a hiba előrejelzésre külön-külön, míg a többváltozós módszerrel a metrikák kombinációinak együttes előrejelző képességét vizsgáltuk.

Ha a metrikák mindegyikét a hibaérzékenység külön mutatójaként használjuk, akkor a C3 a 2. legnagyobb pontossággal (accuracy), a 3. legnagyobb fedéssel (recall), az 5. legnagyobb R^2 értékkel és a 6. legnagyobb precizitással (precision) rendelkezik. Ezek az eredmények nem meglepőek, mivel a C3 a kohéziónak csak bizonyos aspektusait ragadja meg, míg a hibákat más, a kohéziót másképp befolyásoló problémák is okozhatják, amelyeket a C3 önmagában nem ragad meg.

Feltételezésünk szerint a C3 kiegészíti a meglévő strukturális mérőszámokat, ezért annak vizsgálatára, hogy a C3 és a strukturális kohéziós mérőszámok kombinálása javíthatja-e a hibaérzékeny osztályok felismerését, többváltozós logisztikus regressziós elemzést alkalmaztunk. Az eredmények szerint a $C3$ metrikát tartalmazó összes modell a precizitás és fedés értékek tekin-

2. táblázat. A többváltozós logisztikus regresszió eredményei a legnagyobb R^2 értékű kohéziós mérőszámok első tíz párjára vonatkozóan (R^2 értékek szerint rendezve).

Model	Accuracy	Acc. Rank	Precision	Prec. Rank	Recall	Rec. Rank	R^2 Values	C_0	C_1	C_2
C3+LCOM3	66.20	1	63.47	26	76.26	6	0.160	1.384	-3.783	0.060
C3+LCOM1	65.23	2	68.23	27	74.27	10	0.154	1.536	-3.471	0.001
C3+LCOM2	64.88	5	67.54	29	73.49	11	0.151	1.572	-3.486	0.001
C3+LCOM4	64.98	4	66.20	30	75.61	7	0.141	1.594	-4.054	0.078
C3+ICH	63.71	6	64.74	34	74.60	9	0.119	1.710	-3.597	0.006
LCOM4+ICH	63.32	9	72.87	16	65.39	15	0.119	-0.717	0.058	0.006
LCOM3+ICH	63.46	7	72.61	17	65.32	16	0.118	-0.703	0.048	0.003
LCOM1+LCOM3	63.27	10	74.16	12	64.12	21	0.116	-0.611	0.001	0.034
LCOM1+LCOM4	61.90	28	73.05	15	61.41	32	0.114	-0.553	0.001	0.030
LCOM1+Coh	62.34	21	72.44	18	64.28	18	0.113	-0.208	0.001	-0.816

tetében a legjobb 12 modell között van. Ez az eredmény alátámasztja azt az elképzelést, hogy a $C3$ valóban *kiegészíti a strukturális metrikákat*, legalábbis a hiba előrejelzés szempontjából. Hasonlóképpen, a precizitás és a fedés szempontjából legjobb modellek azok, amelyek a fogalmi és strukturális metrikák kombinációjára épülnek, ezek minden esetben felülmúlják az egyetlen metrikán alapuló megfelelő modelleket.

Az eredmények értékeléséhez meg kell említeni, hogy a $C3$ metrika alkalmazhatósága függ az azonosítók megfelelő elnevezési konvencióitól és a forráskódban szereplő releváns megjegyzésektől. Ha ezek hiányoznak, akkor a kohézió bármely aspektusának mérésére az egyetlen megbízható módszer továbbra is a strukturális metrikákon alapuló megközelítés.

Főbb eredmények:

- *$C3$ metrika definíciója és számítása (szerzőtársaim eredménye).*
- *A $C3$ metrika szerepének vizsgálata a hiba előrejelzés folyamataiban, hatásának összehasonlítása a strukturális metrikákkal esettanulmányok segítségével (saját eredményem).*
- *Három szoftver forráskódjának analizálása és strukturális kohézió metrikáinak kiszámítása (saját eredményem).*

Eredmények hatása

A kutatás az addigi irodalomban található módszerektől eltérően új megközelítésben vizsgálja az objektum orientált tervezés során előállt osztályok kohézióját. A forráskód struktúrájára épülő megoldások mellett a jelen kontextusban újnak számító LSI módszeren alapuló eljárás a forráskódban implicit módon kódolt fogalmi információkat használja. A természetes nyelvi feldolgozásban már ismert eljárások intuitív adaptálásával olyan metrika ($C3$) meghatározását végeztük el, amely a forráskódban lévő fogalmi viszonyok alapján méri az egyes osztályok kohézióját. Az esettanulmányok megmutatták, hogy az így definiált metrika új dimenziót ad a kohézió méréséhez, amely mind önmagában, mind más metrikákkal kombinálva jó előrejelzője az osztályok hibára való hajlamosságának.

A kapcsolódó cikkre [6] 202 független hivatkozás történt, melyek kategorizálják, összehasonlítási alapként kezelik, illetve építenek is az eredményeinkre.

2. Fogalmi csatolás hatásanalízishez

A csatolás a kohézió mellet a szoftverek egy másik alapvető minőségi jellemzője, amelynek jelentős hatása van a szoftverek megértésére és karbantarthatóságára. Ez a jellemző az egyes modulok között fennálló kapcsolatok erősségét fejezi ki, és jól használható a szoftveren történő változtatások által érintett modulok azonosításában. A csatolás metrikákat egyebek mellett olyan feladatokban alkalmazzák, mint a hatásanalízis [12], osztályok hibaérzékenységének értékelése [18, 43, 5, 35], vagy a szoftverkomponensek azonosítása [28].

A csatolást több tényező befolyásolja, ezért az különbözőképpen is mérhető. A kutatók számos csatolás metrikát javasoltak, azonban a kutatások [13] azt mutatják, hogy ezen metrikák némelyike a csatolás ugyanazon formáját méri, csak különböző mérési módszereket alkalmazva.

A csatolás metrikák hatásanalízisben történő alkalmazásának vizsgálata azt mutatta, hogy a magas CBO (Coupling Between Object classes) strukturális csatolás metrika értékekkel rendelkező osztályokat nagyobb valószínűséggel érintik az egyes, az adott szoftveren végrehajtott változtatások keresztülívelő hatásai. Briand és munkatársai [12] a csatolás mérőszámok használatát vizsgálták a változtatásokkal nagy valószínűséggel érintett osztályok azonosítására. A vizsgálatok kimutatták, hogy a strukturális mérőszámok használhatók a mögöttes függőségi elemzések fókuszálására, azonban feltártak olyan keresztülívelő hatásokat is, amelyeket a strukturális metrikák alapján azonosított, erősen csatolt osztályok nem vesznek figyelembe.

A csatolás mérésére vonatkozó megközelítésünk azon a feltételezésen alapul, hogy az objektum orientált szoftverrendszerek osztályai több módon is kapcsolódhatnak egymáshoz. A kapcsolatok nyilvánvaló és leginkább vizsgált halmaza az adat- és vezérlési függőségeken alapul. Az ilyen strukturális kapcsolatokon kívül az osztályok fogalmilag is kapcsolatban állhatnak egymással, mivel hozzájárulhatnak a modellezett koncepció megvalósításához. A fogalmi csatolás mérésére az LSI módszert használtuk, akárcsak a fogalmi kohézió metrikák esetében.

A fogalmi csatolás metrikákat a koszinusz hasonlóságra építve metóduspárokra építve határoztuk meg. Két metódus m_k és m_j között $CSM^1(m_k, m_j)$-el jelölt csatolás metrika értéke m_k és m_j közötti koszinusz hasonlóság, amennyiben az pozitív, egyébként 0. A CSM^1 felhasználásával definiálható egy adott m_k metódus és egy c_j osztály csatolás metrikája a metódus és az osztály metódusainak CSM^1 értéke átlagaként (Conceptual Coupling between a Method and a Class, $CCMC(m_k, c_j)$), illetőleg két osztály között, a két osztály metódusainak halmazai között definiált rendezetlen párok csatolás metrikáinak átlagaként (Conceptual Coupling between two Classes, $CCBC$). Az így definiált metrikákra nézve megadtuk egy adott osztály fogalmi csatolás mértékét ($CoCC$), amely az adott osztály más osztályokkal történő $CCBC$ értékének az átlaga. Ha egy $c \in C$ osztály erősen kapcsolódik a rendszer többi osztályához, akkor a $CoCC(c)$ értéknek közelebb kell lennie az egyhez, ami azt jelenti, hogy az osztály metódusai fogalmilag erősen kapcsolódnak a többi osztály metódusaihoz.

Az így definiált metrikák mellett a hatásanalízis gyakorlatában a legerősebb csatolással rendelkező komponenseknek van kiemelt szerepe, ezért definiáltuk a maximális fogalmi csatolás mértékét metódus és osztály között ($CCMC_m$), amely az m_k metódus és c_j osztály metódusai között számolt maximális CSM^1 érték. Ezen metrikák átlagából számolva az osztályok között ($CCBC_m$), illetőleg egy adott osztályra vonatkozólag is ($CoCC_m$) megadtuk a maximális fogalmi hasonlóság definícióját.

Az általunk definiált csatolás metrikák használatát esettanulmányon keresztül vizsgáltuk, amelyben összehasonlítottuk őket számos meglévő, ugyanerre a feladatra használt strukturális csatolás metrikával. Az esettanulmányt hasonló módon terveztük meg, mint Briand és munkatársai [12], ahol a szerzők strukturális csatolás metrikákat használtak az osztályok rangsorolására

egy objektum orientált rendszer hatáselemzése során.

Az általunk vizsgált esettanulmányban a $CCBC$ és $CCBC_m$ metrikákat kilenc létező strukturális csatolás metrikával (PIM, ICP, CBO, MPC, $OCMIC$, DAC, $OCAIC$, $ACMIC$ és $ACAIC$) hasonlítottuk össze annak eldöntése érdekében, hogy alkalmazásukkal a hatáselemzés hatékonyabban, pontosabban végrehajtható-e, vagy sem. Az esettanulmányban a Mozilla v1.6 forráskódját használtuk, mivel az egy nagy, valós szoftverrendszer, és rendelkezésre áll hozzá a változások története is. Az összes strukturális csatolás mértéket, beleértve a csatolás mértékek páronkénti változatait is, az általunk kifejlesztett *Columbus* [3] forráskód-elemző eszköz segítségével számoltuk ki. A fogalmi csatolás mértékeket az IRC^{2}M eszközzel [36] határoztuk meg, amely LSI-alapú elemzést végez.

A 3. táblázatban bemutatott eredmények alapján elmondható, hogy a $CCBC_m$ a legjobb csatolás mérték az osztályok rangsorolásához a hatáselemzés során a precizitás, a fedés és F-mérték szempontjából is. A precizitás és fedés eredményeket tartalmazó táblázat a disszertációban részletesen bemutatásra kerül.

3. táblázat. Hatásanalízis F-mértékei (csökkenő sorrendben).

	10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**200**	**300**	**400**	**500**	**Avg**
CCBC$_m$	19.1	23.3	24.8	24.8	23.9	22.9	21.9	21.1	20.3	19.5	14.5	11.6	9.86	8.62	**19**
ICP	8.77	9.89	10.5	10.8	11.3	11.7	11.3	10.8	10.4	10.1	7.19	5.91	5.34	4.93	**9.22**
PIM	8.35	9.7	10.3	10.7	11.2	11.6	11.2	10.7	10.4	10.1	7.15	5.89	5.33	4.93	**9.12**
CCBC	7.34	9.18	9.38	9.24	9.05	8.94	8.71	8.51	8.39	8.27	6.95	6.24	5.85	5.59	**7.97**
CBO	6.7	6.85	5.92	5.05	4.45	3.95	3.61	3.28	3.01	2.77	1.96	1.9	2.28	2.54	**3.88**
MPC	6.16	4.97	3.99	3.24	2.73	2.36	2.15	1.93	1.74	1.59	1.34	1.52	2.01	2.33	**2.72**
OCMIC	2.06	1.48	1.15	0.92	0.77	0.67	0.68	0.61	0.55	0.5	0.85	1.23	1.8	2.25	**1.11**
OCAIC	1.85	1.32	0.98	0.78	0.65	0.56	0.59	0.53	0.48	0.43	0.82	1.23	1.79	2.24	**1.02**
DAC	1.88	1.34	0.93	0.81	0.67	0.53	0.53	0.53	0.53	0.37	0.35	0.33	0.33	0.32	**0.68**
ACMIC	0.56	0.43	0.35	0.29	0.25	0.23	0.29	0.26	0.24	0.22	0.69	1.16	1.73	2.2	**0.64**
ACAIC	0.43	0.33	0.27	0.23	0.2	0.19	0.26	0.23	0.21	0.2	0.69	1.15	1.73	2.2	**0.6**

Az eredmények azt mutatják, hogy a $CCBC_m$ hatékony indikátora (az általunk vizsgált csatolás mértékek közül a legjobb) az objektum orientált rendszerek osztályai egy külső tulajdonságának - a változásra való hajlamnak. Ez a csatolás mérték hatékonyan használható a releváns osztályok rangsorolására az objektum orientált rendszerek hatáselemzése során.

Az esettanulmányban csak olyan strukturális metrikákat vettünk figyelembe, amelyek a forráskódból nyert statikus információkon alapulnak. A fogalmi csatolás mérőszámok a forráskódban szereplő azonosítók és megjegyzések racionális elnevezési konvencióitól függnek. Amennyiben ezek hiányoznak, a csatolás bármely aspektusának mérésére az egyetlen hatékony módszer a strukturális csatolás méréseken nyugszik.

Főbb eredmények:

- *A fogalmi csatolás metrikák definíciója és számítása (szerzőtársaim eredménye).*
- *A fogalmi metrikák hatásanalízisben történő vizsgálata esettanulmány útján (saját eredményem).*
- *Az esettanulmányhoz használt szoftverrendszer strukturális csatolás metrikáinak kiszámítása statikus forráskódelemzés útján (saját eredményem).*

Eredmények hatása

Egy nagy méretű szoftverrendszer hatékony elemzéséhez a benne lévő osztályok kapcsolatainak, illetve a kapcsolatok erősségének ismerete elengedhetetlen. Számos csatolás metrika ismert a szakirodalomban, azonban tapasztalatok alapján ezek gyakran ugyanazt a lényegi tulajdonságot fejezik ki, csak eltérő formalizmussal. Ebben a munkában új, szemantikai analízisre épülő csatolás metrikákat definiáltunk: $CCBC$ és $CCBC_m$ az osztályok páronkénti, illetőleg $CoCC$ és $CoCC_m$ az osztályok rendszerszintű jellemzésére. Vizsgáltuk a metrikák használhatóságát a hatásanalízis folyamataiban összevetve meglévő kilenc strukturális metrikával. Az eredmények alapján a $CCBC_m$ metrika átlagosan több mint kétszeres javulást jelent a második helyezett strukturális metrikához képest.

A kapcsolódó cikkre [7] 144 független hivatkozás történt, melyek metrikáinkat és eredményeinket új összehasonlítások, illetve új eszközök kiértékelésének alapjaként is felhasználják.

3. Új fogalmi csatolás és kohézió metrikák

Tovább vizsgálva a fogalmi mérőszámokat, két új metrikát definiáltunk $CCBO$ (Conceptual Coupling between Object Classes) és $CLCOM5$ (Conceptual Lack of Cohesion of Methods) néven. Ezek leginkább abban különböznek az eddigi fogalmi metrikáktól, hogy a strukturális csatolás és kohézió metrikák által inspirált számolási mechanizmust alkalmaznak, és a bennük figyelembe vett viszonyok erősségei vágási határértékkel paraméterezhetőek.

A $CLCOM5$ definiálásához először az előző fejezetben bemutatott CSM^1 metrikát tesszük paraméterezhetővé és binárissá (CSM^P), hogy csak egy bizonyos CSM^1 határérték felett jelezzen kapcsolatot két metódus között. Értéke 0 vagy 1 lehet. A $CLCOM5$ értéke egy osztály minden metóduspárjából így létrejövő gráf erősen összefüggő komponenseinek száma.

Hasonlóan, a $CCBO$ definiálásához a $CCBC$ hasonlósági metrikát tesszük paraméterezhetővé és binárissá ($CCBC^P$), hogy csak egy határérték felett jelezzen kapcsolatot. Értéke 0 vagy 1 lehet. A $CCBO$ értéke az adott osztály minden más osztállyal vett $CCBC^P$ metrika értékeinek összege.

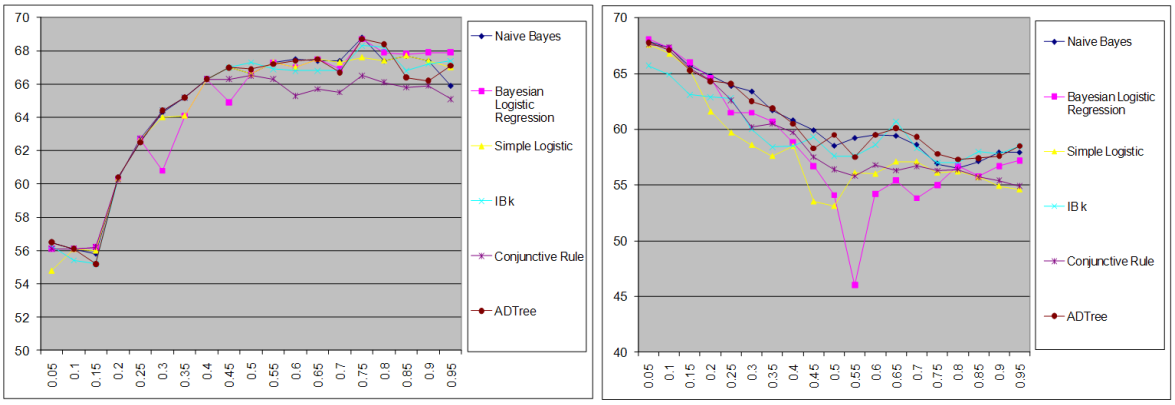
Az új metrikák kiértékeléséhez számos létező fogalmi és strukturális metrikával végeztünk összehasonlítást. A $CLCOM5$ és a $CCBO$ a főkomponens analízis szerint nem alkotnak ugyan külön dimenziót, mint ahogy azt a $C3$ tette korábban, mivel fogalmi jellegük ellenére jobban hasonlítanak a strukturális metrikákra, mint más fogalmi metrikák. A $CLCOM5$ például erős (0.7 feletti együtthatójú) korrelációt mutatott számos strukturális metrikával (LOC, LLOC, NOI, CBO, RFC, WMC). Ez pozitív eredményként értelmezhető, hiszen a fogalmi metrikák hatékonyabban számíthatók és függetlenek a programozási nyelvtől.

4. táblázat. A fogalmi és strukturális metrikák hibajelző képességének kereszt-validációja, szótövezéssel és anélkül.

ML Algorithm	Conceptual-no-stem				All-metrics-no-stem				Conceptual-with-stem				All-metrics-with-stem			
	A	**P**	**R**	**F**	**A**	**P**	**R**	**F**	**A**	**P**	**R**	**F**	**A**	**P**	**R**	**F**
Bayesian Log. Reg.	68.3	70.5	69.1	69.8	71.8	76.7	67.3	71.7	68.8	70.4	71.2	70.8	71.5	71.5	74.7	72.3
Bayes Net	67	64.7	83.4	72.8	70.5	72.5	71.5	72	67.3	65	83.1	73	70.5	72.4	71.8	72.1
Naïve Bayes	68.1	67.9	75.9	71.7	69.2	73.1	66.3	69.6	68.5	67.9	77.2	72.3	69.1	73.1	66.3	69.5
Logistic Regression	67.6	73.6	61	66.7	72.5	76.6	69.5	72.8	67.9	72.6	63.4	67.7	72.4	77	68.4	72.4
RBF Network	67.1	70.1	66.4	68.2	69.3	70.7	71.9	71.3	68.7	68.8	75.1	71.8	69.7	71.9	70.5	71.2
Simple Logistic	67.6	73.5	60.9	66.6	72.1	75.9	69.6	72.6	67.8	72.6	63.4	67.7	71.8	75.6	69.2	72.3
SMO	67.8	73.8	61.1	66.9	72.4	76.2	69.8	72.8	68	72.5	64.1	68	72.2	76	69.7	72.7
IB-k	66.6	67.9	70.3	69.1	71.2	73.1	72.5	72.8	68.8	70.5	70.9	70.7	72.7	74.5	73.8	74.2
Conjunctive Rule	65.8	79.9	47.6	59.6	69.6	81.8	55	65.8	64.5	73.1	52.6	61.2	69.5	82.1	54.3	65.4
Decision Table	67.8	65.8	81.9	73	70.3	73.6	68.5	71	68.1	66.5	80.7	72.9	70.5	74.3	67.9	71
AD Tree	68.4	65.9	84.2	73.9	70.9	72.8	72.3	72.5	68.3	65.7	84.6	74	71	74.4	69.3	71.7
REP Tree	67.3	67.8	73.2	70.4	71.2	72.6	73.6	73.1	67.6	68.5	72.3	70.3	70.6	72.2	72.6	72.4

A fogalmi metrikák megbízhatósága a megjegyzések és azonosítók minőségén felül függhet az LSI által használt korpusz előfeldolgozásától is. Ezért megvizsgáltuk a szótövezés hibajelző képességre kifejtett hatását a fogalmi metrikáknál önálló, illetve strukturális metrikákkal kombinált esetben. Ahogyan a 4. táblázat mutatja, a szótövezés növekedéshez vezetett a megalkotott predikciós modellek pontosságában, fedésében és az F-mértékében. Sőt, ez a pozitív változás következetesen megtalálható minden különböző gépi tanulási algoritmus esetében. Összesítve, a szótövezett fogalmi metrikák és a strukturális metrikák kombinációja vezet a legjobb eredményekhez. Ezek fényében kijelenthetjük, hogy a szótövezés javítja a hiba előrejelzés teljesítményét. Legjobb tudomásunk szerint ez volt az első empirikus vizsgálat, ami a szótövezés szoftverminőséghez köthető hatását elemzi.

1. ábra. CLCOM5 (balról) és CCBO (jobbról) pontosságok különböző határértékeknél.



Mivel a $CCBO$ és a $CLCOM5$ paraméteres mérőszámok, a teljesítményüket nagyban befolyásolhatja a kiszámításukkor használt határérték. Eddig alapértelmezésben 0,7-et használtunk, de ezen felül fontos a (szoftverenként változó) optimális paraméter meghatározása. Megismételtük ezért a hiba előrejelzés pontosságának vizsgálatát az alanyként szolgáló Mozilla rendszerre vetítve minden határértékkel a [0,05–0,95] intervallumban 0,05-ös lépésközt alkalmazva (lásd az 1. ábrát). Eredményeink szerint minden alkalmazott gépi tanulási algoritmus egyetértett abban, hogy a $CLCOM5$ optimuma 0,7 és 0,8 között, míg a $CCBO$-é 0,05 és 0,1 között található. Általánosítva azt a következtetést vonhatjuk le, hogy CLCOM5 esetében a magasabb, $CCBO$ esetében pedig az alacsonyabb határértékek vezetnek jobb előrejelzési teljesítményhez.

5. táblázat. A CCBO és a CLCOM5, valamint a C3 kombinálása a strukturális metrikákkal.

Algorithm	**CCBO, CLCOM5+struct**			**C3+struct**		
	A	**P**	**R**	**A**	**P**	**R**
Bayes. Log. Reg.	71.6	74.8	70.1	71.7	73.1	73.9
Bayes Net	70.7	72.6	72	70.3	72.3	71.6
Naïve Bayes	69.2	73	66.5	68.9	73	65.9
Logistic Reg.	72	75.8	69.6	71.8	76.3	68.1
RBF Network	69.8	72.4	69.8	69.8	72.4	69.5
Simple Logistic	71.6	75.5	69	71.9	75.6	69.5
SMO	72	74	72.7	72.1	75.6	70.1
IB-k	73	75.4	72.9	72.3	74.7	72.2
Conjunctive Rule	69.7	81.1	56	69.9	80.9	56.7
Decision Table	70.4	74.1	67.9	70.4	74.1	67.9
AD Tree	71	72.9	72.3	70.5	74.3	67.9
REP Tree	71.1	73.4	71.3	70.6	72.6	71.6

Végül megvizsgáltuk azt is, hogy a strukturális metrikák hiba előrejelző képességét a $CCBO$ és $CLCOM5$ páros, vagy a korábban definiált $C3$ metrika javítja-e jobban (lásd az 5. táblázatot). A pontosság, precizitás és fedés értékek elemzése alapján azt a következtetést vontuk le, hogy a $CCBO$ és $CLCOM5$ metrikák hozzáadása nagyobb javulást eredményez.

A fenti empirikus vizsgálatok kimutatták tehát, hogy az újonnan definiált $CCBO$ és a $CLCOM5$ metrikák nem csak felhasználhatóak a hiba előrejelzésben, de létező strukturális metrikákkal kombinálva hatékonyan képesek javítani a hiba előrejelző modellek pontosságát.

Főbb eredmények:

- *Két új fogalmi metrika (CCBO, CLCOM5) definíciója (saját eredményem).*
- *Az új metrikák hiba előrejelző képességeinek empirikus vizsgálata, mind önállóan, mind a létező metrikákkal kombinálva (saját eredményem).*
- *A szótövezés előfeldolgozási módszer fogalmi-alapú metrikákra gyakorolt hatásának empirikus vizsgálata (szerzőtársaim eredménye).*

Eredmények hatása

Ezen kutatás során két új fogalmi metrikát (CCBO és CLCOM5) definiáltunk a korábbi fogalmi metrikákat bevezető munkánkra építve. A megalkotott új metrikák hasonlítanak a korábban definiált fogalmi metrikákhoz, ám közelebb állnak a hagyományos, strukturális metrikákhoz és gyorsabban, könnyebben és nyelvfüggetlen módon számíthatók. Empirikus kísérletek során megmutattuk, hogy a strukturális metrikákkal kombinálva jobb eredményt érnek el a programok hiba előrejelzésében gépi tanulás során, mint a korábban definiált fogalmi metrikák.

A kapcsolódó cikk [9] elnyerte az IEEE Source Code Analysis and Manipulation (SCAM) rangos nemzetközi konferencia legjobb cikke díját 2010-ben. A munkára 50 független hivatkozás történt, amelyek többsége előrejelző modellek (hiba, olvashatóság, szoftverevolúció, program alkalmazás terület automatikus becslése) lehetséges prediktoraként használja a definiált metrikákat, de számos szakirodalom összehasonlító mű tárgyát is képezi.

4. Egységesített publikus hibaadatbázis hibák előrejelzéséhez

A szoftverfejlesztés egyik legégetőbb problémája a szoftverekben található hibák felderítése és kijavítása, amely mind a fejlesztő, mind a felhasználó érdeke. A hibák utólagos megkeresése és javítása rendkívül erőforrás-igényes tevékenység. A hiba előrejelzés egy olyan folyamat, amely során a múltban elkövetett hibákat felhasználva próbáljuk megtanulni a hibák jellemzőit egy előrejelző modell segítségével, hogy a lehetséges hibák még fejlesztés közben kijavításra kerülhessenek. Ez napjainkban rendkívül aktívan kutatott terület, amellyel számos kutató foglalkozik. A kutatást élénkítendő számos hiba adatbázis nyíltan elérhető [41, 37, 45, 21, 26, 38]. Ezeknek nagy előnye lehet, hogy egy új előrejelző modell készítéséhez (vagy egy meglévő validálásához) is felhasználhatjuk őket. Azonban a legtöbb adatbázis különböző módon készült, különböző adatokat tartalmaz a hibákról, szoftverekről, különböző metrikákat (független változókat) tárol a hibákról, forráskódokról, így az egyes adatbázisok együttes használata (például mély tanuláshoz) vagy a hiba előrejelző modellek kereszt-validációja a különböző adatbázisokon rendkívül problémás lehet.

Kutatásunk során, mely egy egységes hibaadatbázis előállítását célozta, feltérképeztük és kiértékeltük a kapcsolódó szakirodalmat. Kigyűjtöttük a releváns, elérhető és nyílt forrású adatbázisokat, melyek Java programozási nyelven készült szoftverek hibáira specializálódtak. Ezek a következők: PROMISE – Jureczko [26], Eclipse Bug Dataset [45], Bug Prediction Dataset [15], Bugcatchers Bug Dataset [24], valamint a GitHub Bug Dataset [8]. A rendszerek alapvető statisztikáit a 6. táblázat mutatja be.

6. táblázat. Statisztika a vizsgált hiba adatbázisokról.

Adatbázis	**Rendszerek száma**	**Kódsorok száma**	**Granularitás**
PROMISE	14	2 805 253	Osztály
Eclipse Bug Dataset	1	3 087 826	Fájl
Bug Prediction Dataset	5	1 171 220	Osztály
Bugcatchers Bug Dataset	3	1 833 876	Fájl
GitHub Bug Dataset	15	1 707 446	Osztály, Fájl

A hibaadatbázisok a legtöbbször különböző metrikákat (prediktorokat) tartalmaznak. Habár voltak azonos nevű metrikák is, vizsgálatunk során kiderült, hogy ezek értékei nem szükségszerűen azonos módon kerültek meghatározásra. A probléma kiküszöbölése érdekében letöltöttük az adatbázisokban szereplő minden egyes rendszer adott forráskód-verzióit, és az összes rendszert leelemeztük az általunk fejlesztett forráskód elemzővel, az ingyenes és nyílt forráskódú *OpenStaticAnalyzer* 1.0 (OSA) használatával (ami a Columbus forráskód analizátorunk utódja), hogy egységes forráskód metrikákat kapjunk, melyeket prediktorként használhatunk. Az elemzési folyamat nem volt triviális, hiszen több mint 10 millió kódsor elemzését kellett végrehajtanunk, különböző Java verziót használó projekteken. Az eredményeket felhasználva előállítottunk egy egységes hibaadatbázist osztály- és fájlszinten. Az egyes hibaadatbázisok egységesítése során felmerülő hibákat és megoldásukat az értekezésben részletezzük.

Az egységesített hiba adatbázis tartalmazza az összes adatbázis adatát az eredeti metrikákkal, melyet kiegészítettünk az OSA által számított további metrikákkal. Az így létrejövő adatbázis 47 618 osztály szintű, továbbá 43 744 fájl granularitású adatot tartalmaz. A hiba adatbázisokat

a hozzájuk tartozó metaadatokkal együtt teljes körűen kiértékeltük. A metaadatok elemzése magában foglalja például a használt statikus elemzők, az adatbázisok granularitásának mértékének, a hibakövető és a verziókezelő rendszerek, valamint a használt metrikakészletek vizsgálatát.

Összehasonlítottuk az eredeti metrikák, az egységesített metrikák és a kettő együttes hiba előrejelzési képességeit. Ehhez a Weka [23] J48 (C4.5 döntési fa) algoritmusát használtuk fel minden esetben, mivel a célunk az adatbázisok vizsgálata, nem pedig a legjobban teljesítő gépi tanulási modell kiválasztása volt, a J48 algoritmus pedig már korábbi kutatásunk [8] alkalmával megfelelőnek bizonyult erre a feladatra.

A kutatás során először a projektenkénti tanulást vizsgáltuk meg (egy adott projekten tanul az algoritmus, majd egy eddig ismeretlen hiba is a projektből érkezik), 10-szeres keresztvalidációt használva. Az eredmények vegyesek voltak, míg egyes projektek esetében nagyon magas F-mértéket kaptunk (0,992), más projektek esetében jóval alacsonyabbat (0,655). A konkrét magas F-mérték annak a következménye, hogy a hibák eloszlása ebben az adathalmazban szintén rendkívül magas (98,79%). Az eredmények közti különbségeket megvizsgálva arra jutottunk, hogy az egyes hiba adatbázisok összevonása segíthet egy általánosan jól teljesítő modell megvalósításában.

A kutatás további részében az egységesített nagy hiba adatbázissal dolgoztunk. Az egységesítés segítségével sikerült elfednünk a kisebb adatbázisok hiányosságait. 10-szeres keresztvalidálással értékeltük a J48 által épített hibajelző modellt ezen a nagy adathalmazon is. Az F-mérték 0,818 lett az osztályok esetében és 0,755 a fájlok esetében, míg az AUC 0,721 lett az osztályok esetében és 0,699 a fájlok esetében.

7. táblázat. A legdominánsabb metrikák adatbázisonként.

Dataset	**Level**	**Dominant predictors**
Bug Prediction Dataset	class	**wmc**, **TNOS**, cvsEntropy
GitHub Bug Dataset	class	**WMC**, NOA, **TNOS**
PROMISE	class	LOC, DIT, TNM
Unified Bug Dataset	class	CLOC, TCLOC, CBO, NOI, DIT
Bugcatchers Bug Dataset	file	**code**, PDA, SpeculativeGenerality
GitHub Bug Dataset	file	**McCC**, NumOfPrevMods, NumOfDevCommits
Eclipse Bug Dataset	file	TypeLiteral, NSF_max, **MLOC_sum**
Unified Bug Dataset	file	**LOC**, **McCC**

A modellek osztály- és fájlszintű betanítása után megvizsgáltuk, mely prediktorok a legdominánsabbak. A Weka mind osztály-, mind fájlszinten metszett döntési fákat ad nekünk. A 6. táblázat a legdominánsabb metrikákat mutatja be a konstruált döntési fákból egyes adatbázisokra vonatkozóan. Azokat a metrikákat kiemeltük, amelyek több adatbázisban is előfordulnak. Osztályszinten a WMC (Weighted Methods per Class) és a TNOS (Total Number of Statements) a legfontosabbak, azonban az egységesített hiba adatbázisban történetesen nem ezek voltak a legdominánsabbak. Ami az egységesített hibaadatbázist illeti, a CLOC, TCLOC (comment lines of code), DIT (Depth of Inheritance), CBO (Coupling Between Objects) és NOI (Number of Outgoing Invocations) a legdominánsabb metrikák az osztályok szintjén. Fájlszinten a legdominánsabb prediktorok a különböző adatbázisokban több átfedést mutatnak az egységesített hibaadatbázisban dominánsakkal. A LOC (Lines of Code) és a McCC (McCabe's Cyclomatic Complexity) voltak a legfontosabb változók a döntési fában.

A kutatásunk utolsó lépéseként különböző projekteket használtunk a tanításhoz, valamint a teszteléshez (projektközi tanulás). Ezt a lépést a különböző adatbázisok összes rendszerén elvégez-

tük. Az eredményeink szintén vegyesek voltak, az átlagos F-mértékek 0,4 és 0,6 között mozogtak, azonban előfordultak rendkívül szélsőséges értékek is egy-egy futtatás folyamán (például 0,078 vagy éppen 0,88) olyan esetekben, amikor a tanítás olyan projekten zajlott, aminek hibaadatbázisában nagyon kevés vagy éppen túl sok hibás elem volt. Az F-mérték mellett megvizsgáltuk az AUC értékeket is. Ez esetben úgy találtuk, hogy nem volt olyan modell, ami minden teszthalmazon rendkívül rosszul teljesített volna az AUC tekintetében. Azonban előfordultak szélsőséges 0,000 és 0,900 AUC értékek is, ami alapján úgy véljük, hogy egy modell teljesítménye ebben az esetben erősen függ a tanító és a teszt adatbázistól is.

Az eredményeink alapján elmondható, hogy az egységesített hiba adatbázis használata stabilabb teljesítményt nyújt, mint a korábbi hiba adatbázisok önmagukban.

Főbb eredmények:

- *Módszertan hiba adatbázisok egységesítésére (közös eredmény).*
- *Egységesített hiba adatbázis, ami összesen több, mint 91 ezer bejegyzést tartalmaz (szerzőtársaim eredménye).*
- *Részletes elemzés az egyes hiba adatbázisok metrikáit illetően (szerzőtársaim eredménye).*
- *Empirikus vizsgálat az egységesített hiba adatbázis segítségével előállított gépi tanulási modellek szoftverhiba-előrejelző képességéről (saját eredményem).*

Eredmények hatása

A publikusan elérhető hiba adatbázisok egységesítésével a kutatás új dimenziója nyílhat meg, mivel a jellemzően sok tanuló adatot használó módszerek (mint például a mély neuron hálók) számára korábban nem volt elérhető megfelelő méretű hiba adatbázis, a számos meglévő adatbázis összevonása viszont nem triviális feladat. Az egységesített hiba adatbázis így már lehetőséget kínál a kutatók számára, hogy a fent említett módszereket is bevethessék a hibák előrejelzésében, amelyhez az általunk számított metrikákat is rendelkezésre bocsájtjuk az egyes adatbázisokban megtalálható eredeti metrikák mellett.

Továbbá a kutatási eredmények és módszerek összehasonlíthatóságát is növeli az általunk készített adatbázis. Korábban az egyes publikációk eredményeit nagyon nehézkes volt összehasonlítani, esetleg más adatbázisokból származó adatokon validálni. Azonban az egységesített hiba adatbázis használatával az egyes módszerek és eredmények sokkal objektívebb módon összehasonlíthatóvá válnak. Ezeken felül, a projekteken (és verziókon) átívelő hiba előrejelzés témakörében is kiváló alapokat adhat az általunk készített adatbázis, valamint annak kiértékelése.

Eredményeink fontosságát hangsúlyozza, hogy az azt publikáló cikkünkre [4] már 14 független hivatkozás történt a cikk 2020-as megjelenése óta. Az egységesített hiba adatbázist számos kutató használta munkájában [44, 22, 40], melyek rendre folyóiratokban vagy rangos konferencia kiadványokban jelentek meg.

5. Mélytanulás a hiba előrejelzéshez

A szoftverhibák automatikus detektálásának több módja is létezik, mint például a szabály alapon történő statikus analízis vagy a különböző dinamikus, illetve szimbolikus végrehajtáson alapuló elemzés. Mindezen technikák mellett az utóbbi években a gépi tanulás alapú előrejelző modellek váltak dominánssá. A mesterséges intelligencia területén belül hatalmas lendületet kapott mély tanulást (amely áttöréseket hozott a természetes nyelv feldolgozásban vagy a képfeldolgozásban) azonban eddig csak korlátozott mértékben alkalmazták hiba előrejelzésre. Ennek egyik oka, hogy nehéz a megfelelő méretű és minőségű tanító adathalmaz előállítása, ami szükséges a mély neuronhálók (sok rejtett réteggel rendelkező neurális hálók) betanításához.

Az általunk előállított egységesített publikus hibaadatbázis azonban már alkalmas ilyen vizsgálatok elvégzésére, így megvizsgáltuk a mély neurális hálók hiba előrejelzési képességeit, amit összevetettünk számos más, hagyományos gépi tanuló modell eredményével. A vizsgálat során a forráskódból kinyert statikus metrikák alapján tanítottuk be a modelleket, és elvégeztük a mély neurális hálók optimális hiper-paraméterezésének keresését is. A hatékony tanításhoz szükséges az adatok megfelelő előfeldolgozása, mintavételezése is. A kísérlet elvégzéséhez az egységes hibaadatbázison 10-szeres kereszt-validációt végeztünk el a mély neurális hálók, valamint a következő tradicionális gépi tanuló modellek segítségével: K-legközelebbi szomszéd, Naiv Bayes, döntési fa, véletlen erdő osztályozó, lineáris regresszió, logisztikus regresszió és SVM.

A kereszt-validációhoz a hibaadatbázist 3 részre osztottuk fel 80-10-10% arányban, ahol az első, legnagyobb részhalmaz a tanuló adathalmaz, a két kisebb egy validációs vagy dev adathalmaz és egy kiértékeléshez használható teszt adathalmaz. A validációs adathalmaz segítségével végeztük el a különböző gépi tanuló algoritmusok optimális hiper-paraméterezésének keresését. Mivel a hibaadatbázis kiegyensúlyozatlan, azaz 8 780 (18%) hibás osztályt és 38 838 (82%) hibamenteset tartalmaz, különböző alul- és felül mintavételezési stratégiákat is kipróbáltunk. Továbbá vizsgáltuk a különböző adat előfeldolgozási módszerek hatását is az előrejelzési képességre, mint például az adatok normalizálása, illetve a standardizálás.

A 8. táblázat összefoglalja a legjobb modell eredményeket a kiértékelési, azaz teszt adathalmazon. A legjobb eredmények mintavételezés nélkül, az adatok standardizálásával álltak elő. A táblázat mutatja az egyes algoritmusok legjobb hiper-paraméterezését is. A legjobb mély tanuló háló (CDNNC) 53,59%-os F-mértéket ért el. Az egyetlen teljesítményben azt felülmúló algoritmus a véletlen erdő osztályozó volt 0,12%-os előnnyel F-mértékben.

Érdekes volt megfigyelni, hogy ugyan a két legjobb modell közel azonos eredményt ért el F-mérték szempontjából, a tévesztési mátrixaik lényegesen eltértek. Ez arra enged következtetni, hogy más-más egyedeket osztályoznak jól. Hogy a két leghatékonyabb modell képességeit együttesen is ki tudjuk aknázni, egy egyesített modellt hoztunk létre. Minden egyedre vettük a két modell (mély neurális háló és véletlen erdő) által adott predikciót, és amennyiben a kettő átlaga 0,5 fölé esett, hibásnak jelöltük az egyedet, egyébként hibamentesnek. Az így kapott kombinált modell 55,27%-os F-mértéket (1,56%-os javulás) és 83,99% AUC értéket (1,01%-os javulás) ért el (lásd a 9. táblázatot és a 2. ábrát).

Köztudott, hogy a mély neurális hálók betanításához nagyságrendekkel több adatra van szükség, mint a hagyományos modellek esetén. Ezért, hogy ellenőrizzük, várhatóan kijönne-e a teljesítménybeli különbség a mély neurális hálók és a többi modell között ha nagyságrendekkel több adat állna rendelkezésre a tanításhoz, egy kiegészítő kísérletet végeztünk. Mivel több adatot előállítani nagyon nehéz, inkább csökkentettük a tanító adathalmaz méretét (25%, 50%, és 75%, illetve 100%-át használtuk fel), hogy azon figyeljük meg a modellek teljesítményének tendenciáját a rendelkezésre álló adatok mennyiségének függvényében.

8. táblázat. A tanuló algoritmusok eredményei a legjobb paraméterezéssel.

Alg.	Paraméterek	Teszt		Idő	
		F-mérték	AUC	Tanítás	Futás
erdő	--max-depth 10 --criterion entropy --n-estimators 100	**53.71%**	**82.98%**	87.7s	0.5s
cdnnc	--layers 5 --neurons 250 --batch 100 --lr 0.1	53.59%	81.79%	2132.5s	12.7s
knn	--n_neighbors 18	52.40%	81.14%	124.3s	273.2s
svm	--kernel rbf --C 2.6 --gamma 0.02	52.25%	70.75%	3142.0s	106.2s
fa	--max-depth 10	49.77%	77.34%	11.1s	0.1s
logistikus	--penalty l2 --solver liblinear --C 2.0 --tol 0.0001	46.43%	78.06%	58.4s	0.1s
lineáris		45.61%	77.47%	3.9s	0.1s
bayes		34.84%	74.40%	0.5s	0.1s

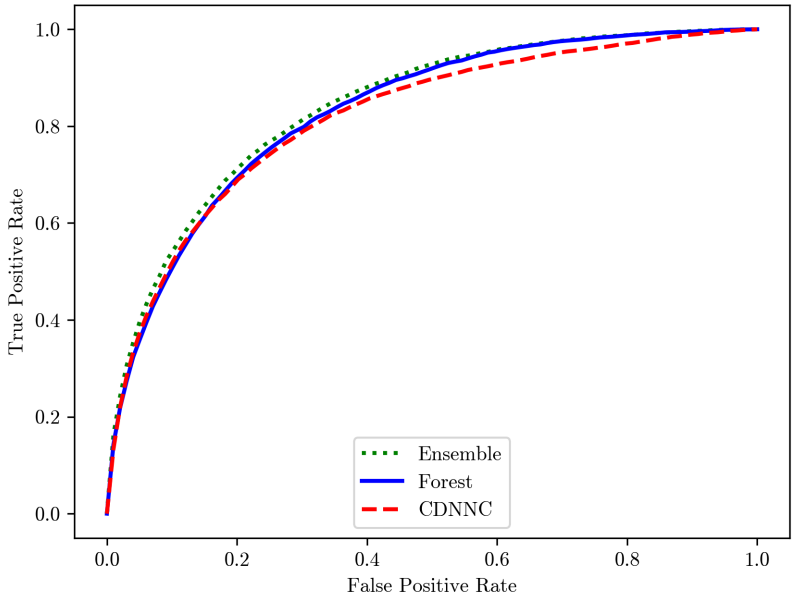
9. táblázat. Az egyéni és a kombinált eredmények összehasonlítása.

	CDNNC	Erdő	Kombinált
Precízió	47.24%	49.97%	**50.49%**
Fedés	61.90%	58.06%	**61.05%**
F-Mérték	53.59%	53.71%	**55.27%**
AUC	81.79%	82.98%	**83.99%**

A méréseink szerint a mély neurális háló gyorsabban javul az adatmennyiség növekedésével, mint a többi algoritmus, így hipotézisünk nagy valószínűséggel igaz, miszerint egyre több tanító adat esetén a mély tanulás egyértelműen hatékonyabb előrejelzést tesz majd lehetővé (a részleteket lásd az értekezésben).

Főbb eredmények:

- *Részletes módszertan mély neurális hálók hiba előrejelzésben történő alkalmazására és optimális hiperparamétereinek megtalálására (saját eredményem).*
- *Mély neuronháló felépítése a hibák előrejelzésére, annak kiértékelése és összehasonlítása a hagyományos módszerekkel (saját eredményem).*
- *Kombinált modell kialakítása, amely a mély neurális hálók és a véletlen erdő módszerek ötvözésével a többi modelltől jobb előrejelzési pontosságot eredményez (közös eredmény).*
- *Empirikus kísérlet, mely alátámasztja, hogy az elérhető tanító adatok mennyiségének növekedésével a mély neurális hálók előrejelzési pontossága várhatóan túlszárnyalja a többi módszert (közös eredmény).*



2. ábra. A CDNNC, erdő és a kombinált ROC összehasonlítása.

Eredmények hatása

Noha sem a mély tanulás, sem pedig a hiba előrejelzés önmagukban nem újszerű témák, mégis a kettő ötvözéséről nagyon kevés ismeretünk van. Munkánkban egy olyan részletes módszertant mutatunk be lépésről lépésre, amely megkönnyíti a gyakorlatban is használható mély neurális hálók építését és finomhangolását, amelyek elérik, illetve elegendő tanító adat esetén meg is haladják a legjobb korábbi hiba előrejelző modellek teljesítményét.

A módszer és közölt eredmények hatását jól szemlélteti, hogy az azt publikáló cikkünkre [2] annak ellenére máris számos hivatkozás történt, hogy az csak két és fél éve, 2020 nyarán jelent meg. A 11 külföldi független hivatkozás nagy része rangos konferencián megjelenő cikkből, illetve a szakterületen magas presztízzsel bíró impakt faktoros folyóiratból érkezett, mint például a Journal of Systems and Software [19], Information and Software Technology [42] vagy az Empirical Software Engineering [29].

A munkánk hatását tovább erősíti, hogy az olyan részletességű, ami lehetővé teszi a közvetlen gyakorlati alkalmazását. Ennek támogatására nem csak az empirikus kísérletünk eredményeit tettük közzé, de megvalósítottuk a modellek előállításához, tanításához és kiértékeléséhez is használható szoftver keretrendszert, amelyet nyílt forrású projektként elérhetővé is tettünk (`https://github.com/sed-inf-u-szeged/DeepBugHunter`). Módszerünk és a publikált szoftvereszköz együttes hatásaként mind a hiba előrejelzés gyakorlati adaptációjának, mind pedig a területen végzett további kutatások fellendülését várjuk.

Hivatkozások

A szerző hivatkozott publikációi

[1] Tibor Bakota, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A probabilistic software quality model. In *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM 2011)*, pages 243–252, Williamsburg, VA, USA, September 2011. IEEE Computer Society.

[2] Rudolf Ferenc, Dénes Bán, Tamás Grósz, and Tibor Gyimóthy. Deep learning in static, metric-based bug prediction. *Array*, 6:100021, July 2020. Open Access.

[3] Rudolf Ferenc, Árpád Beszédes, Mikko Tarkiainen, and Tibor Gyimóthy. Columbus – reverse engineering tool and schema for C++. In *Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002)*, pages 172–181, Montréal, Canada, October 2002. IEEE Computer Society.

[4] Rudolf Ferenc, Zoltán Tóth, Gergely Ladányi, István Siket, and Tibor Gyimóthy. A public unified bug dataset for java and its assessment regarding metrics and bug prediction. *Software Quality Journal*, 28:1447–1506, 2020. Open Access.

[5] Tibor Gyimóthy, Rudolf Ferenc, and István Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, November 2005.

[6] Andrian Marcus, Denys Poshyvanyk, and Rudolf Ferenc. Using the conceptual cohesion of classes for fault prediction in object oriented systems. *IEEE Transactions on Software Engineering*, 34(2):287–300, March 2008.

[7] Denys Poshyvanyk, Andrian Marcus, Rudolf Ferenc, and Tibor Gyimóthy. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering*, 14(1):5–32, February 2009.

[8] Zoltán Tóth, Péter Gyimesi, and Rudolf Ferenc. A public bug database of GitHub projects and its application in bug prediction. In *Proceedings of the 16th International Conference on Computational Science and Its Applications (ICCSA 2016)*, pages 625–638, Beijing, China, July 2016. Springer International Publishing.

[9] Béla Újházi, Rudolf Ferenc, Denys Poshyvanyk, and Tibor Gyimóthy. New conceptual coupling and cohesion metrics for object-oriented systems. In *Proceedings of the 10th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2010)*, pages 33–42, Timişoara, Romania, September 2010. IEEE Computer Society. Best paper of the conference.

További hivatkozások

[10] J. M. Bieman and B-K. Kang. Cohesion and reuse in an object-oriented system. In *Proceedings of the 1995 Symposium on Software Reusability*, SSR '95, pages 259–262. ACM, 1995.

[11] L. C. Briand, J. W. Daly, and J. Wüst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 3:65–117, 1998.

[12] L. C. Briand, J. Wust, and H. Lounis. Using coupling measurement for impact analysis in object-oriented systems. In *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, pages 475–482, Aug 1999.

[13] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, 51(3):245–273, may 2000.

[14] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, jun 1994.

[15] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *7th Working Conference on Mining Software Repositories (MSR)*, pages 31–41. IEEE, 2010.

[16] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, sep 1990.

[17] M. Dowson. The ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes*, 22(2):84, 1997.

[18] K. El Emam, W. Melo, and J. C. Machado. The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software*, 56(1):63–75, feb 2001.

[19] Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, and Bedir Tekinerdogan. On the use of deep learning in software defect prediction. *Journal of Systems and Software*, 195:111537, 2023.

[20] R. L. Glass. Frequently forgotten fundamental facts about software engineering. *IEEE software*, 18(3):112–111, 2001.

[21] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. Reflections on the nasa mdp data sets. *IET software*, 6(6):549–558, 2012.

[22] Tao Hai, Jincheng Zhou, Ning Li, Sanjiv Kumar Jain, Shweta Agrawal, and Imed Ben Dhaou. Cloud-based bug tracking software defects analysis using deep learning. *Journal of Cloud Computing*, 11(1):1–14, 2022.

[23] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[24] T. Hall, M. Zhang, D. Bowes, and Y. Sun. Some code smells have a significant but small effect on faults. *ACM Trans. Softw. Eng. Methodol.*, 23(4):33:1–33:39, September 2014.

[25] R. Hiesgen, M. Nawrocki, T. C. Schmidt, and M. Wählisch. The race to the vulnerable: Measuring the log4j shell incident. *arXiv preprint arXiv:2205.02544*, 2022.

[26] M. Jureczko and L. Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, pages 9:1–9:10, New York, NY, USA, 2010. ACM.

[27] S. Kyatam, A. Alhayajneh, and T. Hayajneh. Heartbleed attacks implementation and vulnerability. In *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–6. IEEE, 2017.

[28] J. K. Lee, S. J. Jung, S. D. Kim, W. H. Jang, and D. H. Ham. Component identification method with coupling and cohesion. In *Proceedings Eighth Asia-Pacific Software Engineering Conference*, pages 79–86, Dec 2001.

[29] Francesco Lomio, Sergio Moreschini, and Valentina Lenarduzzi. A machine and deep learning analysis among sonarqube rules, product, and process metrics for fault prediction. *Empirical Software Engineering*, 27(7):189, 2022.

[30] R. F. E. Lorch Jr and E. J. O'Brien. *Sources of coherence in reading.* Lawrence Erlbaum Associates, Inc, 1995.

[31] A. Marcus. *A Semantic Driven Program Analysis.* PhD thesis, Kent State University, Kent, OH, USA, 2003.

[32] A. Marcus and D. Poshyvanyk. The conceptual cohesion of classes. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 133–142, Sep. 2005.

[33] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *11th Working Conference on Reverse Engineering*, pages 214–223, Nov 2004.

[34] T. M. Meyers and D. Binkley. Slice-based cohesion metrics and software intervention. In *11th Working Conference on Reverse Engineering*, pages 256–265, Nov 2004.

[35] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on Software Engineering*, 33(6):402–419, jun 2007.

[36] D. Poshyvanyk and A. Marcus. The conceptual coupling metrics for object-oriented systems. In *2006 22nd IEEE International Conference on Software Maintenance*. IEEE, sep 2006.

[37] G. Robles. Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 171–180. IEEE, 2010.

[38] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215, 2013.

[39] R. Shetty, K-K. R. Choo, and R. Kaufman. Shellshock vulnerability exploitation and mitigation: a demonstration. In *International Conference on Applications and Techniques in Cyber Security and Intelligence: Applications and Techniques in Cyber Security and Intelligence*, pages 338–350. Springer, 2018.

[40] Alexander Trautsch, Steffen Herbold, and Jens Grabowski. A longitudinal study of static analysis warning evolution and the effects of pmd on software quality in apache open source projects. *Empirical Software Engineering*, 25(6):5137–5192, 2020.

[41] E. J. Weyuker, R. M. Bell, and T. J. Ostrand. Replicate, replicate, replicate. In *Replication in Empirical Software Engineering Research (RESER), 2011 Second International Workshop on*, pages 71–77. IEEE, 2011.

[42] Xi Xiao, Yuqing Pan, Bin Zhang, Guangwu Hu, Qing Li, and Runiu Lu. Albfl: A novel neural ranking model for software fault localization via combining static and dynamic features. *Information and Software Technology*, 139:106653, 2021.

[43] P. Yu, T. Systa, and H. Muller. Predicting fault-proneness using oo metrics. an industrial case study. In *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, pages 99–107, March 2002.

[44] Yanyang Zhao, Yawen Wang, Yuwei Zhang, Dalin Zhang, Yunzhan Gong, and Dahai Jin. St-tlf: Cross-version defect prediction framework based transfer learning. *Information and Software Technology*, 149:106939, 2022.

[45] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Proceedings of the third international workshop on predictor models in software engineering*, page 9. IEEE Computer Society, 2007.