Újszerű hiba-előrejelző adatbázis, folyamatmetrikák és egy publikus JavaScript hibaadatbázis

Gyimesi Péter

Szoftverfejlesztés Tanszék
Szegedi Tudományegyetem

Szeged, 2023

Témavezető:

Dr. habil. Ferenc Rudolf

Ph.D. értekezés tézisei

Szegedi Tudományegyetem
Informatika Doktori Iskola

Bevezetés

A szoftverhibák menedzselése esszenciális része a szoftverfejlesztésnek és a cégek hajlamosak jelentős erőforrást áldozni erre. A programozók annak ellenére is hajlamosak hibát véteni, hogy egyre több és jobb eszköz áll rendelkezésükre az automatikus hibadetektálásra [24]. A szoftverhibák kezelése olyan feladatokból áll, mint a hibamegelőzés, hibakeresés és hibajavítás.

A szoftverhibák keresésének folyamata általában abból áll, hogy a hibabejelentések alapján manuálisan ellenőrzik a kódot és megkeresik a probléma gyökerét. Ez idő- és erőforrás-igényes tevékenység, és a szükséges erőfeszítés minimalizálása elősegítené a fejlesztési költségek csökkentését. Az egységtesztek is segíthetnek szoftverhibákat detektálni és lokalizálni, de ehhez az szükséges, hogy teszteseteket írjunk a fejlesztéssel párhuzamosan, ami szintén egy erőforrásigényes feladat. A hibalokalizáció elősegítésének egy másik módja, hogy jellemezzük az ismert hibákat néhány megfelelő mérőszámmal és megpróbáljuk előrejelezni, hogy mely forráskódelemek tartalmaznak a legnagyobb valószínűséggel hibát. A legfontosabb lépés a hibadetektálás megkönnyítéséhez, hogy elemezzük a már ismert hibákat, ezáltal beazonosítsunk mintákat vagy trendeket.

Az ismert hibák elemzéséhez szükség van a forráskód változástörténetére és egy hibakövető rendszerre. Manapság számos fejlesztő használ verziókezelő rendszert - úgy mint a Subversion vagy Git -, ezáltal a forráskód változástörténete gyakran rendelkezésre áll. A hibakövető rendszerek használata szintén elég általános a szoftverfejlesztésben. Számos kereskedelmi és nyílt forráskódú szoftverrendszer elérhető ezekre a célokra. A hibabejelentések rögzítésre kerülnek ezekben a rendszerekben és minden, a hibához kapcsolódó változást is nyomon követnek, beleértve a forráskódjavításokat. Továbbá különféle webes szolgáltatások is készültek ezeknek az igényeknek a kielégítésére. A legnépszerűbbek, úgy mint a SourceForge, a Bitbucket és a GitHub betöltik a fent említett funkciókat. Általában számos szolgáltatást nyújtanak, ilyenek a forráskódhosting és a felhasználókezelés. Az API-jaik lehetővé teszik, hogy kinyerjünk különféle típusú adatot, például támogatást nyújtanak a felhasználók viselkedésének vagy együttműködésének vizsgálatára, vagy akár magának a forráskódnak az elemzéséhez. Mivel ezeknek a rendszereknek a többsége tartalmazza a hibakövetést, felvetődött az ötlet, hogy ezeket az információkat a hibás forráskódrészek jellemzésére használják fel [38]. Ehhez a hibabejelentéseket össze kell kapcsolni a megfelelő forráskódrészekkel. Általános gyakorlat a verziókövető rendszerekben, hogy leírják a változásokat egy commithoz tartozó megjegyzésben és gyakran megadják a társított hibabejelentés azonosítóját is, amit a commit hivatott javítani [26]. Ez felhasználható a forráskód hibás verzióinak azonosítására. A GitHub több mint 330 millió tárolót tartalmaz, és egy könnyen használható API-val rendelkezik ezekhez a projektekhez, amelyek a Git-en keresztül érhetőek el; ezért kézenfekvő választás a vizsgálatok adatforrásaként.

Ami a programozási nyelveket illeti, a ma használt legnépszerűbb nyelvek közé tartozik a Java, a Python, a C++, a JavaScript és a PHP. A TIOBE Index szerint 2023-ban a legnépszerűbb programozási nyelv a Python volt, ezt követte a C, a C++ és a Java. A Java sok éve népszerű programozási nyelv — skálázhatósága, megbízhatósága és hordozhatósága miatt széles körben használják a vállalati szoftverfejlesztésben. A JavaScript (JS) a de facto webes programozási nyelv világszerte, és a GitHubon leginkább elterjedt nyelv. A JavaScript-et széles körben használják a webalkalmazások kliens oldalán a magas reagálóképesség és felhasználóbarátság elérése érdekében. Az elmúlt években rugalmasságának és hatékonyságának köszönhetően egyre gyakrabban alkalmazták szerveroldali fejlesztésekhez is, ami full-stack webalkalmazásokhoz vezetett [11]. Az olyan platformok, mint a Node.js, lehetővé teszik a fejlesztők számára, hogy kényelmesen fejleszthessék az alkalmazásokat teljes egészében JavaScript-ben. Népszerűsége ellenére a JavaScript belső jellemzői — mint például a gyenge típusosság, a prototípusöröklődés és a futásidejű kiértékelés — az egyik leginkább hibára hajlamos programozási nyelvvé teszik.

Mint ilyen, a szoftvermérnöki kutatások nagy része a JavaScript webalkalmazások elemzésére és tesztelésére összpontosított [14, 35, 36, 29, 20, 9, 28, 10]. Ezen okok miatt, bár sokféle programozási nyelv létezik, tudatosan úgy döntöttünk, hogy erőfeszítéseinket a Java és JavaScript projektekben jelentett hibák elemzésére összpontosítjuk.

Bár hatalmas mennyiségű nyers adat áll rendelkezésre a szoftverhibákkal kapcsolatban, az összegyűjtés, szűrés és feldolgozás idő- és erőforrás-igényes feladat lehet. Számos cikk foglalkozott a hibaadatbázisokkal, sokféle megközelítést alkalmazva, mint például hiba-előrejelzés, hibalokalizálás vagy tesztelési technikák [17, 32, 30, 27, 39]. A kutatók gyakran használnak saját céljaikra létrehozott adatbázist, de ezeket az adathalmazokat ritkán teszik közzé a közösség számára. A szoftverhibákkal kapcsolatos kutatások bősége ellenére a nyilvánosság számára hozzáférhető hibaadatbázisok száma feltűnően kevés. Továbbá az elemzett programokat vagy a mellékelt kísérleti adatokat ritkán teszik elérhetővé részletesen, leíró jelleggel, gondozott és koherens módon. Ez nem csupán akadályozza a tanulmányok reprodukálhatóságát, hanem meg is nehezíti a kutatók számára, hogy felmérjék a legkorszerűbb kapcsolódó kutatásokat és összehasonlítsák a létező megoldásokat. Pontosabban, a tesztelési technikákat jellemzően a meglévő programok hibáinak detektálásában mutatott hatékonyságuk alapján értékelik ki, azonban a valódi hibákat nehéz elkülöníteni, reprodukálni és jellemezni. Ebből kifolyólag az általános gyakorlat a manuálisan hozzáadott hibákon vagy mutációs tesztelésen alapul [23]. Ezen megoldások mindegyikének vannak korlátai. A manuálisan hozzáadott hibák a kutatók elvárásai felé torzulhatnak, aláásva az ezeket használó tanulmányok reprezentativitását. A mutációs technikák viszont lehetővé teszik mesterséges hibák nagy számú generálását. Bár a kutatások kimutatták, hogy a mutánsok meglehetősen reprezentatívak a valódi hibákra nézve [21, 25, 12], a mutációs tesztelés számítási szempontból költséges a gyakorlatban. Ezen okokból kifolyólag a nyilvánosan elérhető hibaadatbázisok kiemelkedően fontosak az újszerű hibakeresési, hiba-előrejelzési, hibalokalizációs vagy programjavítási megközelítések kidolgozásához.

A hibás forráskódelemek különféle módszerekkel történő jellemzése továbbra is népszerű kutatási terület. Az ismeretlen hibás kódelemek automatikus felismeréséhez előfeltétel a már ismertek jellemzése. Számos jó tanulmány elérhető a hibajellemzéssel kapcsolatban [22, 34, 16, 19]. Egy hibát javító commit módosításának feldolgozása segíthet a hiba által érintett pontos kódrészletek meghatározásában. A hibajellemzésre leggyakrabban használt módszerek között szerepel a szövegbeli hasonlóság hibás kódrészekkel [13], forráskódelemzés, termékmetrikák [18, 33] vagy folyamatmetrikák. Számos olyan eszköz van, némelyik ingyenes, amelyek képesek különféle nyelven írt forráskódok elemzésére és termékmetrikák előállítására. Vizsgálataink során az OpenStaticAnalyzer eszközt használtuk forráskódelemzésre, mert képes akár Java, akár JavaScript programozási nyelven írt forráskód feldolgozására, és részletes információkat tud kinyerni a forráskód elemeiről.

Munkánk három tézispontból áll. Ezen tézispontok céljai a következők:

I. Egy újszerű módszer bemutatása hibaadatbázis építésére, és annak a hiba-előrejelzésére gyakorolt hasznosságának kiértékelése egy hagyományos megközelítéssel készült adatbázissal összevetve.

II. Egy módszer bemutatása a szoftver-folyamatmetrikák gráfadatbázis segítségével történő kiszámítására, a metrikák hiba-előrejelző erejének kiértékelése, és azok összehasonlítása a termékmetrikákkal.

III. Egy manuálisan ellenőrzött, valódi JavaScript hibákat tartalmazó adatbázis bemutatása, és ezen hibák mennyiségi és minőségi elemzéseinek bemutatása.

I. Újszerű predikciós hibaadatbázis és kiértékelése

A korábban közzétett adathalmazok hagyományos megközelítést követve készültek a hiba-előrejelzési technikák tesztelésére. Ezek az adathalmazok jellemzően tartalmazzák az összes forráskódelemet, mind a hibásakat és a nem hibásakat, az elemzett rendszer egy vagy több verziójából. Ebben a tézispontban egy új megközelítést mutatunk be, amely a hibák által érintett forráskódelemek pillanatképeinek összegyűjtésére összpontosít, azok jellemzőivel együtt a hibák kijavítása előtt és után. Ez a megközelítés kizárja azokat a kódelemeket, amelyeket nem érintettek ismert hibák.

Az ilyen típusú adathalmaz felhasználásával hatékonyan megfigyelhetjük a termékmetrikák változásait a hibajavítások során. Ez lehetővé teszi számunkra, hogy megvizsgáljuk a hibás és nem hibás kódelemek forráskódmetrikái közötti különbségeket, és értékes betekintést nyerjünk. Elemzésünk elvégzéséhez kiválasztottunk 15 nyílt forráskódú projektet a GitHubról és figyelembe vettük az összes bejelentett hibát a hibakövető rendszerükből. Három forráskódszinten építettünk adatbázisokat: fájl, osztály és metódus. Ennek az új adatbázisnak a neve *BugHunter Dataset*.

11 gépi tanulási algoritmust használva előrejelzési modelleket építettünk, és demonstráltuk az adatbázisban rejlő lehetőségeket a hiba-előrejelzésre és további vizsgálatokra. Fontos vizsgálati szempont, hogy az újszerű adatbázisból épített hiba-előrejelzési modellek hogyan viszonyulnak a hagyományos adatbázisból készült modellekhez. Ezen összehasonlítás elvégzéséhez felhasználtuk a korábbi hagyományos adatbázisunkat, mely ugyanazon 15 projekt felhasználásával készült és *GitHub Bug Dataset* [7] néven hivatkoztunk rá.

A következő kutatási kérdéseket vizsgáltuk:

- **Kutatási kérdés 1:** Használható a *BugHunter Dataset* hiba-előrejelzésre?
- **Kutatási kérdés 2:** Jobb előrejelző képességűek az osztályszintre vetített metódusszintű metrikák, mint maguk az osztályszintű metrikák?
- **Kutatási kérdés 3:** Erőteljesebb és kifejezőbb a *BugHunter Dataset*, mint a *GitHub Bug Dataset*?

Az *első kutatási kérdés* megválaszolásához elemeztük a különböző gépi tanulási algoritmusok által adott eredményeket metódus-, osztály- és fájlszinten. A kapott eredményeket figyelembe véve kijelenthetjük, hogy a metódusszintű hiba-előrejelzési modellek sikeresebbek, mint a fájl- és osztályszintűek a teljes adathalmazt tekintve. Továbbá megfigyeltünk eltéréseket az F-measure értékek között a különböző projektekben, ami alátámasztja a hipotézisünket, miszerint nem minden projekt ad alkalmas tanítóhalmazt. Ígéretes F-measure értékeket kaptunk egyes projekteken, elérve a 0,7573-at metódusszinten, 0,7400-at osztályszinten és 0,7741-et fájlszinten. Erre a kutatási kérdésre pozitív választ tudunk adni és azt mondhatjuk, hogy a létrehozott adathalmaz alkalmas hibapredikcióra.

Az adatbázis metódus- és osztályszinten egyaránt tartalmaz hibainformációt, és ismerjük az osztályok és metódusok közötti tartalmazási kapcsolatot is. Mivel azonban az osztályok eltérő forráskódmetrikákkal rendelkeznek, mint a metódusok, felmerült egy kérdés: használhatjuk-e (és ami még fontosabb, ajánlott-e) a metódusszintű metrikákat hibás osztályok előrejelzésére? A *második kutatási kérdés* megválaszolásához végrehajtottunk egy kísérletet, ahol a metódusszintű tanulási eredményeket osztályszintre vetítettük. A metódusszintű tanulási eredmények keresztvalidációja során a metódusokat tartalmazó osztályok segítségével a hibás (legalább egy hibás metódust tartalmazó) és nem hibás osztályokból számoltuk ki az igazságmátrixot.

Ezt az eredményt összehasonlítottuk az osztályszintű előrejelzés eredményével. Az eredmények az 1. táblázatban azt mutatják, hogy a vetítéses módszer sokkal jobban teljesít, mint az osztályszintű metrikákkal történő előrejelzés.

1. táblázat. A vetítéses tanulás eredményei

Algoritmus	**Precision**		**Recall**		**F-Measure**	
	Vetített	Osztály	Vetített	Osztály	Vetített	Osztály
trees.RandomForest	0,7471	0,5336	0,7370	0,5336	0,7405	0,5334
trees.RandomTree	0,7421	0,5381	0,7273	0,5380	0,7330	0,5376
functions.SGD	0,7441	0,5718	0,7288	0,5676	0,7322	0,5626
rules.DecisionTable	0,7425	0,5703	0,7404	0,5705	0,7309	0,5637
trees.J48	0,7390	0,5531	0,7250	0,5530	0,7290	0,5520

Az osztályszintű hiba-előrejelzés metódusszintű metrikákkal teljesített a legjobban a tanulmányunkban; az F-measure értékek elérték a 0,74-et. Gyanítjuk, hogy ennek az osztályszintű metrikák általánossága az oka, amelyek ennél fogva nem elég erőteljesek ahhoz, hogy hatékonyan felismerjék a forráskódhibákat. Az első kutatási kérdésre adott válasz kiterjesztéseként, a fent leírt mechanizmust adhatjuk az osztályszintű hibák nagyobb pontosságú megtalálásához egy szoftverrendszerben.

A különböző adathalmazok kifejező erejének összehasonlítása kemény feladat, mivel a különféle adathalmazok eltérő céllal készültek, ezért gyakran csak néhány független változó közös bennük. A *harmadik kutatási kérdés* megválaszolásához egy objektív összehasonlítást adunk a hagyományos adatbázisunk, a *GitHub Bug Dataset* és a *BugHunter Dataset* között. Az adathalmazok pontosan ugyanazt a 15 projektet foglalják magukba, és a független változóik halmaza közös, illetve azonos módon, azonos eszközzel számoltuk őket. Ugyanazokat a gépi tanulási algoritmusokat használtuk az előrejelző modellek építéséhez. Ily módon meglehetősen magától értetődő, hogy összehasonlítsuk a kifejezőképességét és a kompaktságát a két adathalmaznak.

2. táblázat. Az adathalmazok előrejelző képessége és mérete

Adathalmaz	**Átlag**	**Szórás**	**Min**	**Max**	**Méret**
	Metódusszint				
BugHunter	0,6319	0,0836	0,3376	0,7573	109 244
Hagyományos	0,7348	0,0789	0,4019	0,8339	167 708
	Osztályszint				
BugHunter	0,5685	0,0704	0,3572	0,7400	66 092
Hagyományos	0,7710	0,0869	0,3446	0,8331	27 216
	Fájlszint				
BugHunter	0,5147	0,0749	0,3328	0,7741	49 868
Hagyományos	0,6058	0,1076	0,2882	0,8247	16 235
	Vetített				
BugHunter	0,7405	0,0914	0,3178	0,8386	-
Hagyományos	0,7831	0,0716	0,4399	0,8825	-

Először is összehasonlítjuk az adatbázisok méretét a bennük található bejegyzések száma alapján (lásd 2. táblázat). Metódusszinten *1,54*-es, osztályszinten *0,41*-es és fájlszinten *0,33*-as arányt kaptunk. A metódusszinten kapott arány magasabb, mint 1,0, ami azt mutatja, hogy az új megközelítés kevesebb bejegyzést tartalmaz ezen a szinten. Azt várnánk, hogy az új módszer a hagyományosnál kevesebb bejegyzést tartalmaz, mivel a *BugHunter Dataset* csak olyan bejegyzéseket tartalmaz, amelyeket érintett egy javított hiba. Azonban a hagyományos *GitHub Bug Dataset* csupán a benne található projektek méretétől függ (fájlok, osztályok és metódusok száma). Ezzel ellentétben a *BugHunter Dataset* nagyban függ a rendszerekben található lezárt hibabejelentések számától (egy nagy projektben előfordulhat, hogy kevés bejelentett hiba van). Összességében nem tudjuk egyértelműen eldönteni, hogy az újszerű adathalmaz kompaktabb lenne-e, azonban az tisztán látszik, hogy metódusszinten tömörítené a hibákkal kapcsolatos információkat. A metódusszinten elért F-measure érték 0,6319 és a tömörített adathalmaz a hagyományosnál 58 464-gyel kevesebb bejegyzést tartalmaz. Mindkét adathalmazban a bejegyzések száma elegendő előrejelző modellek felépítéséhez, azonban először az előrejelző képességeket kell megvizsgáljuk, mielőtt következtetéseket vonnánk le a kifejezőerőt és a tömörséget illetően.

A 2. táblázat bemutatja a metódus-, osztály-, és fájlszinten kapott gépi tanulási eredményeket, illetve a vetítéses módszerrel kapott F-measure értékeket.

A hagyományos *GitHub Bug Dataset*-en a gépi tanulási algoritmusok jobban teljesítettek, magasabb F-measure értékeket elérve minden esetben. A kétféle adathalmaz alapvetően különbözik, mivel két különböző módszerrel készültek. A hagyományos adathalmaz építésének módszere némi bizonytalansághoz vezet az adathalmazban, mert megtörténhet, hogy a hiba még nincs jelen a hozzárendelt főverzióban. A 3. táblázat bemutatja néhány jellemzőjét ennek a bizonytalanságnak.

3. táblázat. Bizonytalanság a hagyományos adathalmazban

Projekt	**Napok átlaga**	**Commitok számának átlaga bejelentés előtt**	**Commitok számának átlaga javítás előtt**
Android U. I. L.	78,78	179,04	22,82
ANTLR v4	83,73	94,83	66,21
Broadleaf Comm.	96,40	524,88	116,74
Eclipse Ceylon	136,05	442,00	20,22
Elasticsearch	93,85	1 004,60	382,79
Hazelcast	84,61	1 905,88	143,54
jUnit	91,94	76,71	171,09
MapDB	102,09	150,47	25,06
mcMMO	108,71	289,83	41,72
Mission Control T.	64,00	203,00	55,93
Neo4j	39,53	535,77	189,30
Netty	83,65	411,60	48,96
OrientDB	99,21	568,76	179,30
Oryx	63,00	104,42	3,40
Titan	51,35	65,91	59,85

A második oszlop a főverzió kiadása és a hiba bejelentése között eltelt napok számának át-

laga. Láthatjuk, hogy ezek az átlagok meglehetősen magasak; a teljes átlag 85 nap. A harmadik oszlop a főverzió kiadása és a hiba bejelentése között a projekten történt commitok számának átlaga. Ezek az értékek projektenként változnak, mivel a fejlesztők tevékenységi szintjétől függenek. Néhány projektnél (Elasticsearch, Hazelcast) ez jelenthet több ezer módosítást a hiba bejelentése előtt. Minél több commitot hajtottak végre, annál magasabb a valószínűsége, hogy a forráskódelem a főverzió kiadása után lett hibás. A negyedik oszlop mutatja a hiba bejelentése és a hiba kijavítása között végrehajtott commitok számának átlagát. Ezek a számok sokkal kisebbek, ami azt is mutatja, hogy a hibákat viszonylag gyorsan javították. Ez a gyors korrekciós magatartás azt okozza, hogy a javítás előtti és utáni állapotok kevésbé különböznek egymástól a BugHunter módszerben. Következésképpen a metrikaértékek kisebb eltérése nehezebbé teszi a pontos előrejelző modellek építését.

Az új BugHunter módszer viszont mentes a fentebb említett bizonytalanságtól, mert a hibákhoz kapcsolódó forráskódelemeknek csak a hibás és a teljesen kijavított állapotait használja. Ily módon az előállított adathalmaz sokkal precízebb, tehát sokkal alkalmasabb a gépi tanulásra. Következésképpen a magasabb F-measure értékek ellenére sem állíthatjuk egyértelműen, hogy a hagyományos adathalmaz jobb lenne. A két adathalmaz értékei közötti eltérés 0,10 metódusszinten, 0,21 osztályszinten, és 0,09 fájlszinten. A metódusszintű metrikák osztályszintre történő vetítése majdnem olyan magas F-measure értéket ért el (0,7405), mint a hagyományos esetben (0,7831). A különbség mindössze 0,04, mégis egy sokkal pontosabb adathalmazon.

A szerző hozzájárulása

A szerző számos jelentős hozzájárulást tett ehhez a kutatáshoz. Először is ő tervezte meg és implementálta a hibaadatbázis készítésére bemutatott újszerű módszert. Továbbá aktívan részt vett a szakirodalom átnézésében és a BugHunter adathalmazba történő bekerülési feltételek definiálásában. A szerző volt felelős a módszer végrehajtásáért, az adathalmaz előállításáért, és a projektekkel kapcsolatos statisztikák összegyűjtéséért. Végül, a szerző vezető szerepet vállalt a tanulmány során használt gépi tanulási technikák eredményeinek előállításában és elemzésében. A tézispont a következő publikációkra épül:

- **Péter Gyimesi**, Gábor Gyimesi, Zoltán Tóth, and Rudolf Ferenc. Characterization of Source Code Defects by Data Mining Conducted on GitHub. In *15th International Conference on Computational Science and Its Applications (ICCSA 2015), Banff, AB, Canada, June 22–25*, pages 47–62, LNCS, Volume 9159. Springer International Publishing, 2015.

- Zoltán Tóth, **Péter Gyimesi**, and Rudolf Ferenc. A Public Bug Database of GitHub Projects and its Application in Bug Prediction. In *16th International Conference on Computational Science and Its Applications (ICCSA 2016), Beijing, China, July 4–7*, pages 625–638, LNCS, Volume 9789. Springer International Publishing, 2016.

- Rudolf Ferenc, **Péter Gyimesi**, Gábor Gyimesi, Zoltán Tóth, and Tibor Gyimóthy. An Automatically Created Novel Bug Dataset and its Validation in Bug Prediction. Journal of Systems and Software, 2020, 169: 110691.

II. Folyamatmetrikák számítása és hiba-előrejelző képességük

Tanulmányok kimutatták, hogy a folyamatmetrikák felülmúlják a termékmetrikákat a hiba-előrejelzésben. Azonban kevés a kutatás ezen a területen, és a folyamatmetrikák használata korlátozott maradt. Következésképpen ebben a tézispontban ezeket a hiányosságokat célozzuk meg egy hatékony módszer bemutatásával a különböző folyamatmetrikák számolására.

22 folyamatmetrika számítását implementáltuk fájlokra, osztályokra és metódusokra, együtt a megfeleltetett hibaszámokkal. A folyamatmetrikák hatékonyságának kiértékeléséhez kiválasztottunk öt nyílt forráskódú Java projektet a GitHubról, és adatbázisokat generáltunk minden projekt 5 főverziójára. Végül kiértékeltük a hiba-előrejelző képességeiket, és összehasonlítottuk őket a termékmetrikákkal.

A következő kutatási kérdéseket vizsgáltuk:

- **Kutatási kérdés 1:** Használható-e a folyamatmetrikákat tartalmazó adathalmaz hiba-előrejelzési célokra?
- **Kutatási kérdés 2:** Mely metrikák hatékonyabbak szoftverhibák előrejelzésére: a folyamatmetrikák vagy a termékmetrikák?
- **Kutatási kérdés 3:** Milyen kapcsolat van a termékmetrikák és a folyamatmetrikák között?

Az *első kutatási kérdés* megválaszolásához kiértékeltük a 11 gépi tanuló algoritmust mind a 25 főverzión csak a folyamatmetrikák használatával. A kapott eredmények azt sugallják, hogy a folyamatmetrikákat tartalmazó adathalmaz alkalmas hiba-előrejelzési célokra. Legjobban a RandomForest és a DecisionTable algoritmusok teljesítettek. Konkrétan az elért F-measure értékek 0,7997 (Titan) osztályszinten, 0,8180 (MapDB) fájlszinten, és 0,8185 (MapDB) metódusszinten a RandomForest algoritmus használatával. Továbbá fontos megjegyezni, hogy nem minden projekt egyformán képes megfelelő tanítóhalmazt nyújtani a hiba-előrejelzéshez. Ezen felül átlagban a folyamatmetrikák erősebben indikálják a hibahajlamosságot a forráskód magasabb szintjein, úgy mint osztályok vagy fájlok.

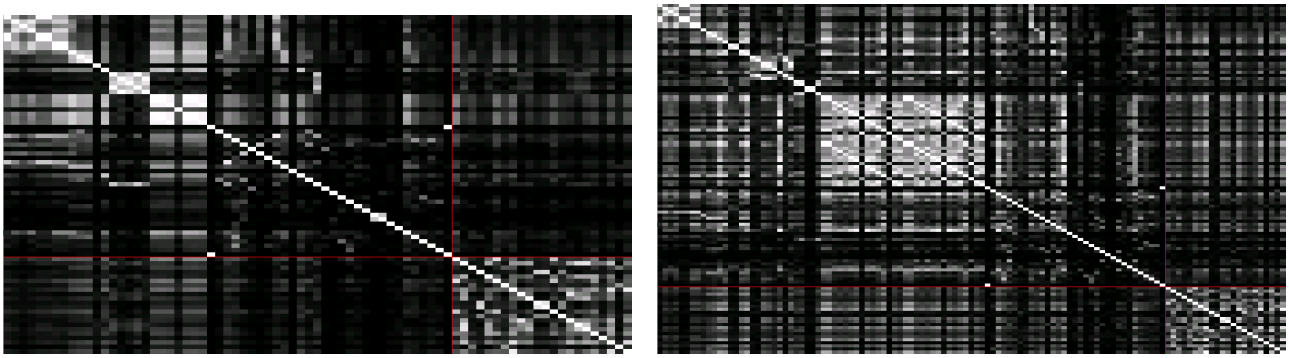
4. táblázat. Termékmetrikákkal és folyamatmetrikákkal elért átlagos F-measure értékek összehasonlítása

	Fájl		**Osztály**		**Metódus**	
	Termék	*Folyamat*	*Termék*	*Folyamat*	*Termék*	*Folyamat*
ANTLR4	**0,7035**	0,6940	**0,7129**	0,6748	**0,7198**	0,5142
Broadleaf C.	**0,6926**	0,6766	**0,7470**	0,6772	**0,7830**	0,6116
MapDB	0,6420	**0,7860**	**0,6939**	0,6531	0,6709	**0,7328**
jUnit	0,5647	**0,6805**	**0,7164**	0,6143	**0,5824**	0,5350
Titan	0,5759	**0,6968**	0,6971	**0,7386**	0,6252	**0,7004**

A termék- és folyamatmetrikák előrejelző erejének összehasonlításához és a *második kutatási kérdés* megválaszolásához kiértékeltük a két különböző metrikahalmazt minden egyes projektben a legtöbb hibát tartalmazó főverzión. Az elért F-measure értékek a 4. táblázatban kerültek listázásra. Az eredmények alapján arra lehet következtetni, hogy a folyamatmetrikákból fájlszinten épített hiba-előrejelző modellek általánosságban jobb eredményt hoznak a termékmetrikákkal

összehasonlítva. Viszont a forráskód alacsonyabb szintjein, úgy mint az osztály- és metódusszint, a termékmetrikák gyakran felülmúlják a folyamatmetrikákat. Érdekes megfigyelés, hogy a kapott F-measure értékek szórása konzisztensen alacsonyabb a folyamatmetrikákkal szinte minden esetben, átlagosan 0,0375 különbséggel. Ráadásul az összességében elért legalacsonyabb F-measure érték magasabb a folyamatmetrikákkal szinte minden esetben, átlagosan 0,0809 eltéréssel. Ezek a megállapítások azt sugallják, hogy a folyamatmetrikákkal kapott hiba-előrejelzési eredmények robusztusabbak és megbízhatóbbak, jelezve, hogy alkalmasabbak a forráskód hibáinak előrejelzésére.

Végül, a *harmadik kutatási kérdés* megválaszolásához kiszámítottuk a Pearson korrelációs együttható értékeit a termék- és folyamatmetrikák között. Az eredmények azt mutatják, hogy a folyamatmetrikák jobban korrelálnak saját magukkal, mint a termékmetrikákkal. Vannak magasabb korrelációs értékek (~0,45) néhány méret alapú termékmetrika és a folyamatmetrikák között, de általánosságban nincs domináns korrelációs érték a két metrikahalmaz között.



(a) Metódusszint (b) Osztályszint

1. ábra. A termék- és folyamatmetrikák korrelációja

A korrelációs mátrixokat az 1. ábrán illusztráljuk. A fekete cellák jelölik az abszolút értékben alacsony korrelációs értékeket (nullához közeli), miközben a fehér cellák jelölik az abszolút értékben magas korrelációs értékeket (egyhez vagy mínusz egyhez közeli). A folyamatmetrikák a jobb alsó negyedben találhatóak. Kijelenthetjük, hogy a folyamatmetrikák a termékmetrikákkal összehasonlítva eltérő perspektívát nyújtanak a forráskódelemek jellemzéséhez, hiszen nincs erős korreláció a két metrikatípus között.

A szerző hozzájárulása

Ez a tanulmány a szerző önálló munkája. Ő dolgozta fel a szakirodalmat, tervezte meg a módszert, és implementálta a szükséges eszközöket. Azután ő hozta létre a folyamatmetrikákat tartalmazó hibaadatbázisokat, végezte el a kiértékeléseket, és vonta le a következtetéseket. A tézispont a következő publikációkra épül:

- **Péter Gyimesi**. Automatic Calculation of Process Metrics and their Bug Prediction Capabilities. In *Acta Cybernetica*, pages 537–559, Volume 23, No 2, 2017.
- **Péter Gyimesi**. An open-source solution for automatic bug database creation. In *Proceedings of the 10th International Conference on Applied Informatics (ICAI 2017), Eger, Hungary, January 30–February 1*, pages 111–119, 2017.

III. Publikus JavaScript hibaadatbázis

A kiterjedt JavaScript (JS) kutatások ellenére egy jól szervezett, címkézett JS hibákat tartalmazó adattár még mindig hiányzott. A számos JS megvalósítás jelenléte tovább bonyolította az egyesített hibaadatbázis létrehozásának feladatát. Ennek a hiánynak a megszüntetéséhez ebben a tézispontban bemutatunk egy adatbázist, a BugsJS-t, ami magába foglal összesen 453 manuálisan kiválasztott és ellenőrzött hibát 10 nyílt forráskódú JS projektből. Továbbá fejlesztettünk egy keretrendszert az adatbázisunkat felhasználó kutatási folyamatok automatizálásához. Vizsgálatot végeztünk annak megállapítására, hogy a JS hibák hibajavítási mintái összhangban vannak-e meglévő osztályozási sémákkal. A hibák mennyiségi és minőségi elemzése végett létrehoztunk egy taxonómiát az adatbázisban jelenlévő JS hibákból, amely tudomásunk szerint első a maga nemében. Megvizsgáltuk a kategorizálás és a hibajavítási minták közötti kapcsolatot is.

A következő kutatási kérdéseket vizsgáltuk:

- **Kutatási kérdés 1:** A JS hibák hibajavítási mintái a BugsJS-ben megfeleltethetőek-e meglévő osztályozási sémákkal?
- **Kutatási kérdés 2:** Hogyan viszonyulnak a BugsJS hibajavítási mintái a taxonómiánkhoz?

Az *első kutatási kérdés* megválaszolásához megvizsgáltuk a JS hibák hibajavítási mintái és a meglévő osztályozási sémák kapcsolatát. Hasonló tanulmányok [31, 37, 15] korábban már feltártak mintákat Java programok hibajavító módosításaiban, azt sugallva, hogy az efféle minták megléte felfed bizonyos kódkonstrukciókat, amelyek jelezhetnek gyenge pontokat a forráskódban, ahol a fejlesztők következetesen nagyobb eséllyel vétenek hibákat. A Pan és mtsai. [31] által javasolt kategóriákat használtuk a hibajavítási minták osztályozására, melyek eredetileg Java hibajavításokkal kapcsolatosak. A cél az volt, hogy felmérjük, ezek a kategóriák általánosíthatóak-e JS-re, vagy vannak-e konkrétan JS-ben felbukkanó hibajavítási minták. Megállapításaink az 5. táblázatban kerültek listázásra.

5. táblázat. A BugsJS-ben található hibajavító változástípusok

	Kategória	**Példa**	**#**
Létező	`if`-related	`if` feltétel módosítása	291
	Assignments	Értékadás jobb oldalának módosítása	166
	Function calls	Paraméter hozzáadása vagy módosítása	152
	Class fields	Osztály adattag hozzáadása vagy eltávolítása	22
	Function declarations	Függvény szignatúrájának módosítása	94
	Sequences	Függvényhívás hozzáadása hívássorozathoz, ugyanazzal a címzettel	42
	Loops	Ciklus predikátumának módosítása	5
	`switch` blocks	Switch ág hozzáadása vagy eltávolítása	6
	`try` blocks	Egy új `try-catch` blokk bevezetése	1
Új	`return` statements	`return` utasítás módosítása	40
	Variable declaration	Létező változó deklarálása	2
	Initialization	Változó inicializálása üres objektummal vagy tömbbel	3

Az elemzésünk felfedte, hogy az adatbázisunk hibáinak 88%-a beleesik legalább az egyik javasolt kategóriába. A leggyakoribb javítási minták magukba foglaltak `if` utasítás módosítást. Érdekes módon, ugyanazt a három kategóriát találták a leggyakoribbnak Java kódban is. Továbbá beazonosítottunk három JS specifikus ismétlődő mintát.

A hibák mennyiségi és minőségi elemzése végett létrehoztunk egy taxonómiát az adatbázisban jelenlévő JS hibákból. A *második kutatási kérdés* megválaszolásához összevetettük a taxonómiát a hibák javítására használt hibajavítási mintákkal. Az elemzésünket a három fő hibakategóriára összpontosítottuk: incomplete feature implementation, incorrect feature implementation, és generic.

6. táblázat. Taxonómia és hibajavítási típusok

	AS	CF	IF	JS	LP	MC	MD	None	SQ	SW	TY
INCOMPLETE FEATURE IMPLEMENTATION											
configuration processing	1	0	9	1	0	2	1	1	1	0	0
missing type check	1	0	0	0	0	1	0	0	0	0	0
error handling	1	0	12	1	0	3	4	3	1	0	1
callbacks	1	0	4	0	0	2	0	0	1	0	0
incomplete data processing	4	2	9	1	0	11	7	3	1	1	0
incomplete output message	0	0	1	0	0	3	0	0	0	0	0
missing input validation	29	8	53	4	1	11	22	2	2	2	0
empty input parameters	1	0	3	0	0	1	0	0	1	0	0
missing handling of spaces	2	0	2	1	0	1	2	0	0	0	0
missing handling of special characters	6	0	5	1	0	1	0	0	0	0	0
missing null check	0	0	5	2	0	0	0	0	0	0	0
missing type check	18	2	39	2	1	10	10	2	1	1	0
INCORRECT FEATURE IMPLEMENTATION											
configuration processing	5	0	3	1	1	2	1	3	0	0	0
incorrect data processing	24	2	39	5	1	27	13	3	9	1	0
incorrect initialization	5	0	0	0	0	4	0	3	5	0	0
incorrect type comparison	0	0	1	0	0	0	0	0	0	0	0
incorrect filepath	4	0	4	2	0	4	1	0	2	0	0
incorrect handling of regex expressions	13	0	4	0	0	6	3	0	0	0	0
incorrect input validation	25	1	56	12	0	27	15	2	4	1	0
empty input parameters	0	0	1	0	0	0	0	1	0	0	0
incorrect handling of special characters	9	0	7	0	1	12	9	1	3	0	0
unnecessary type check	5	0	7	0	0	0	2	0	0	0	0
incorrect output	1	0	0	3	0	5	0	1	0	0	0
incorrect output message	2	2	8	1	0	10	2	3	0	0	0
performance	1	0	5	0	0	1	0	0	7	0	0
GENERIC											
data processing	1	0	1	2	0	2	0	1	1	0	0
loop statement											
incorrect loop statement	0	0	0	0	0	0	0	1	0	0	0
missing type conversion	2	0	1	0	0	0	0	0	1	0	0
return statement											
incorrect return statement	0	0	0	0	0	1	0	0	0	0	0
missing return statement	0	0	1	0	0	0	0	2	0	0	0
typo	1	0	2	0	0	2	1	0	0	0	0
variable initialization											
incorrect variable initialization	3	1	0	0	0	0	0	0	0	0	0
missing variable initialization	0	4	0	5	0	2	0	0	2	0	0
PERFECTIVE MAINTENANCE	1	0	9	1	0	1	1	0	0	0	0

A 6. táblázat statisztikát szolgáltat az egyes hibajavítási típusok előfordulásáról a hibatípusoknak megfeleltetve. A táblázat a hibajavítási típusok és a hibatípusok között jelentkező összefüggések elemzésére szolgálhat. Összességében véve a leggyakoribb hibajavítások a BugsJS-ben `if`-el kapcsolatosak (291), a második leggyakoribbak értékadással kapcsolatosak (166), és a harmadik leggyakoribbak metódushívással kapcsolatosak (152). Ezek a hibajavítási típusok többnyire összefüggenek a legkiemelkedőbb hibakategóriákkal, ezek nevezetesen: missing input validation, incorrect input validation, és incorrect data processing. Egy másik összefüggés, hogy a reguláris kifejezések javításának preferált módja az értékadással kapcsolatos javítás. Adatbázisunkban a hibajavítási típusok és a hibatípusok között ezek az összefüggések figyelhetők meg.

Ez az átfogó elemzés szemlélteti, hogy az adathalmaz hibák széles skáláját tartalmazza és megbízható adatbázisként szolgálhat reprodukálható kutatások lefolytatásához a szoftverelemzés és tesztelés területén.

A szerző hozzájárulása

A kutatás során a szerző aktívan részt vett a kutatási terv kigondolásában és a keretrendszer implementálásában, mely tartalmazza az adatbázist. Ő felelt a JavaScript projektek gyűjtéséért és elemzéséért, az alkalmas hibák kiválasztásáért és a releváns hibajavítások kinyeréséért. Továbbá részt vett a hibák manuális ellenőrzésében is. A hibák elemzése során a szerző aktívan hozzájárult a hibajavítási minták vizsgálatához, valamint a hibataxonómia létrehozásához és ellenőrzéséhez. Végezetül a szerző vezető szerepet vállalt a hibataxonómia és a hibajavítási minták közötti összefüggések elemzésében. A tézispont a következő publikációkra épül:

- **Péter Gyimesi**, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc and Ali Mesbah. BugsJS: A Benchmark of JavaScript Bugs. In *12th IEEE Conference on Software Testing, Validation and Verification (IEEE ICST 2019), Xi'an, China, April 22–27*, pages 90–101, IEEE, Volume 1. IEEE Computer Society Press, 2019.

- **Péter Gyimesi**, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc and Ali Mesbah. BugsJS: A Benchmark and Taxonomy of JavaScript Bugs. Journal of Software Testing, Verification and Reliability (STVR 2021), John Wiley & Sons Publishing. 38 pages.

- Béla Vancsics, **Péter Gyimesi**, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc and Ali Mesbah. Poster: Supporting JavaScript Experimentation with BugsJS. In *12th IEEE Conference on Software Testing, Validation and Verification (IEEE ICST 2019), Poster Track, Xi'an, China, April 22–27*, pages 375–378, IEEE, Volume 1. IEEE Computer Society Press, 2019.

Összefoglalás

Megmutattuk, hogy korábbi, hagyományos megközelítéssel létrehozott adathalmazok bizonytalanságokat tartalmaznak. Következésképpen kidolgoztunk egy újszerű módszert a hibás forráskódelemek javítás előtti és javítás utáni állapotainak rögzítésére, amely egy csökkentett bizonytalansággal rendelkező adathalmazt eredményezett. Empirikus kiértékeléseket végeztünk és azt találtuk, hogy ez az adathalmaz hasznos lehet hiba-előrejelzésre. Ezenfelül végeztünk egy kísérletet az osztályszintre vetített metódusszintű metrikák hiba-előrejelző képességeinek összehasonlítására az osztályszintű metrikákéval. A megállapításaink azt mutatják, hogy a metódusszintű metrikák osztályszintre vetítése növeli az előrejelző erejüket hiba-előrejelzésben.

Annak ellenére, hogy a korábbi tanulmányok azt mutatták, hogy hiba-előrejelzésben a folyamatmetrikák felülmúlják a termékmetrikákat, a folyamatmetrikák használata nem elterjedt, és kevés a kutatás ezen a területen. Ennek az orvoslására kidolgoztunk egy módszert fájlok, osztályok és metódusok folyamatmetrikáinak hatékony számítására egy gráfadatbázis felhasználásával. Megerősítettük, hogy a hibaadatbázisok folyamatmetrikákkal alkalmasak hiba-előrejelzésre Ezenfelül összevetettük a folyamatmetrikákat a termékmetrikákkal és azt találtuk, hogy a folyamatmetrikák használata hiba-előrejelzésre stabilabb eredményeket hoz. Továbbá a folyamatmetrikák a termékmetrikákkal összehasonlítva eltérő perspektívát nyújtanak a forráskódelemek jellemzéséhez.

Végül létrehoztunk egy adatbázist valódi, manuálisan ellenőrzött JavaScript hibákból, illetve egy keretrendszert, hogy automatizálja a kutatási folyamatokat. Az adatbázis hibáit elemeztük és azt találtuk, hogy a legtöbb rendelkezik olyan javítással, amely beleesik valamelyik létező hibajavítási mintába. Ezenfelül létrehoztunk egy hibataxonómiát és megvizsgáltuk, hogy ez a kategorizálás összefügg-e a hibajavítási mintákkal. Ez az adatbázis egy megbízható forrásként szolgál reprodukálható kutatások elvégzéséhez a szoftverelemzés és tesztelés területén.

A tézispontokat és a kapcsolódó publikációkat a 7. táblázat összegzi.

№	[4]	[7]	[1]	[2]	[3]	[5]	[6]	[8]
I.	◆	◆	◆					
II.				◆	◆			
III.						◆	◆	◆

7. táblázat. A tézispontokhoz kapcsolódó publikációk

Köszönetnyilvánítás

Segítség nélkül nem tudtam volna eljutni idáig, és szeretném kifejezni hálámat a támogatásért, amit kaptam. Végtelenül hálás vagyok a témavezetőmnek, Dr. Ferenc Rudolfnak a felbecsülhetetlen útmutatásáért és tanácsaiért. Megmutatta nekem, hogy elhivatottsággal és kitartással leküzdhetek bármilyen kihívást és elérhetem a céljaimat. A mentorálása rendkívül értékes volt, és hálás vagyok a belém vetett hitért és a számomra biztosított lehetőségekért. Ezenfelül szeretnék

őszinte köszönetet mondani Dr. Gyimóthy Tibornak, a Szoftverfejlesztés Tanszék korábbi vezetőjének, hogy támogatott a PhD-hez vezető utamon. Külön elismerés illeti a munkatársaimat, Dr. Tóth Zoltánt, Vancsics Bélát, és Gyimesi Gábort, akiknek a hozzájárulása és kitartó támogatása fontos szerepet játszott az előrehaladásomban. Továbbá szeretném kifejezni mély elismerésemet minden társszerzőmnek az értékes hozzájárulásáért, illetve Szűcs Editnek a disszertációhoz nyújtott stilisztikai és nyelvtani megjegyzésekért nyújtott segítségéért.

Gyimesi Péter, 2023

Hivatkozások

A szerző megegyező publikációi

[1] Rudolf Ferenc, Péter Gyimesi, Gábor Gyimesi, Zoltán Tóth, and Tibor Gyimóthy. An automatically created novel bug dataset and its validation in bug prediction. *Journal of Systems and Software*, 169:110691, 2020.

[2] Péter Gyimesi. Automatic calculation of process metrics and their bug prediction capabilities. *Acta Cybernetica*, 23(2):537–559, 2017.

[3] Péter Gyimesi. An open-source solution for automatic bug database creation. In *Proceedings of the 10th International Conference on Applied Informatics*, page 111–119, 2017.

[4] Péter Gyimesi, Gábor Gyimesi, Zoltán Tóth, and Rudolf Ferenc. Characterization of source code defects by data mining conducted on github. In *International Conference on Computational Science and Its Applications*, pages 47–62. Springer, 2015.

[5] Péter Gyimesi, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc, and Ali Mesbah. Bugsjs: A benchmark of javascript bugs. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 90–101. IEEE, 2019.

[6] Péter Gyimesi, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Árpád Beszédes, Rudolf Ferenc, and Ali Mesbah. Bugsjs: a benchmark and taxonomy of javascript bugs. *Software Testing, Verification And Reliability*, 31(4):e1751, 2021.

[7] Zoltán Tóth, Péter Gyimesi, and Rudolf Ferenc. A public bug database of github projects and its application in bug prediction. In *International Conference on Computational Science and Its Applications*, pages 625–638. Springer, 2016.

[8] Béla Vancsics, Péter Gyimesi, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc, and Ali Mesbah. Poster: Supporting javascript experimentation with bugsjs. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 375–378. IEEE, 2019.

További publikációk

[9] Christoffer Quist Adamsen, Anders Møller, Rezwana Karim, Manu Sridharan, Frank Tip, and Koushik Sen. Repairing event race errors by controlling nondeterminism. In *Proc. of 39th International Conference on Software Engineering (ICSE)*, 2017.

[10] Saba Alimadadi, Ali Mesbah, and Karthik Pattabiraman. Hybrid DOM-sensitive change impact analysis for JavaScript. In *Proc. of European Conference on Object-Oriented Programming (ECOOP)*, 2015.

[11] Saba Alimadadi, Ali Mesbah, and Karthik Pattabiraman. Understanding asynchronous interactions in full-stack JavaScript. In *Proc. of 38th International Conference on Software Engineering (ICSE)*, 2016.

[12] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proc. of International Conference on Software Engineering*, 2005.

[13] P. Bangcharoensap, A. Ihara, Y. Kamei, and K. Matsumoto. Locating source code to be fixed based on initial bug reports - a case study on the eclipse project. In *Empirical Software Engineering in Practice (IWESEP), 2012 Fourth International Workshop on*, pages 10–15, Oct 2012.

[14] Marina Billes, Anders Møller, and Michael Pradel. Systematic black-box analysis of collaborative web applications. In *Proc. of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2017.

[15] E. C. Campos and M. d. A. Maia. Common bug-fix patterns: A large-scale observational study. In *Proc. of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017.

[16] Cagatay Catal. Software fault prediction: A literature review and current trends. *Expert systems with applications*, 38(4):4626–4636, 2011.

[17] Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[18] C. Couto, C. Silva, M.T. Valente, R. Bigonha, and N. Anquetil. Uncovering causal relationships between software metrics and bugs. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 223–232, March 2012.

[19] Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)*, pages 31 – 41, 2010.

[20] Markus Ermuth and Michael Pradel. Monkey see, monkey do: Effective generation of GUI tests with inferred macro events. In *Proc. of 25th International Symposium on Software Testing and Analysis (ISSTA)*, 2016.

[21] R. Gopinath, C. Jensen, and A. Groce. Mutations: How close are they to real faults? In *Proc. of International Symposium on Software Reliability Engineering*, 2014.

[22] Todd L Graves, Alan F Karr, James S Marron, and Harvey Siy. Predicting fault incidence using software change history. *IEEE Transactions on software engineering*, 26(7):653–661, 2000.

[23] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *Transactions on Software Engineering*, 37(5), 2011.

[24] Bryant Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 672–681. IEEE, 2013.

[25] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *Proc. of ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2014.

[26] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. *MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101, 2014.

[27] Yan Ma, Lan Guo, and Bojan Cukic. A statistical framework for the prediction of fault-proneness. *Advances in Machine Learning Application in Software Engineering, Idea Group Inc*, pages 237–265, 2006.

[28] Magnus Madsen, Frank Tip, Esben Andreasen, Koushik Sen, and Anders Møller. Feedback-directed instrumentation for deployed JavaScript applications. In *Proc. of 38th International Conference on Software Engineering (ICSE)*, 2016.

[29] F. S. Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah. A Study of Causes and Consequences of Client-Side JavaScript Bugs. *IEEE Transactions on Software Engineering*, 43(2):128–144, February 2017.

[30] Thomas J Ostrand, Elaine J Weyuker, and Robert M Bell. Predicting the location and number of faults in large software systems. *Software Engineering, IEEE Transactions on*, 31(4):340–355, 2005.

[31] Kai Pan, Sunghun Kim, and E. James Whitehead. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14(3):286–315, June 2009.

[32] Adam Porter, Richard W Selby, et al. Empirically guided software development using metric-based classification trees. *Software, IEEE*, 7(2):46–54, 1990.

[33] Danijel Radjenović, Marjan Heričko, Richard Torkar, and Aleš Živkovič. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418, 2013.

[34] Emad Shihab, Zhen Ming Jiang, Walid M Ibrahim, Bram Adams, and Ahmed E Hassan. Understanding the impact of code and process metrics on post-release defects: a case study on the Eclipse project. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 4. ACM, 2010.

[35] J. Wang, W. Dou, C. Gao, Y. Gao, and J. Wei. Context-based event trace reduction in client-side JavaScript applications. In *Proc. of International Conference on Software Testing, Verification and Validation (ICST)*, 2018.

[36] J. Wang, W. Dou, Y. Gao, C. Gao, F. Qin, K. Yin, and J. Wei. A comprehensive study on real world concurrency bugs in Node.js. In *Proc. of International Conference on Automated Software Engineering*, 2017.

[37] Hao Zhong and Zhendong Su. An empirical study on real bug fixes. In *Proc. of 37th International Conference on Software Engineering (ICSE)*, pages 913–923, 2015.

[38] Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. *Software Engineering (ICSE), 2012 34th International Conference on*, 2012.

[39] Yuming Zhou and Hareton Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *Software Engineering, IEEE Transactions on*, 32(10):771–789, 2006.

Declaration

In the PhD dissertation of **Péter Gyimesi** entitled *A Novel Bug Prediction Dataset, Process Metrics and A Public Dataset of JavaScript Bugs*, **Péter Gyimesi**'s contribution was decisive in the following results:

1. A Novel Bug Prediction Dataset and its Validation

- Designing the novel bug prediction dataset [3]
- Implementing the designed tools [1], [2], [3]
- Constructing the dataset and gathering statistics [1], [2], [3]
- Analyzing the result of the study [3]

2. Calculation of Process Metrics and their Bug Prediction Capabilities

- Reviewing the existing related studies [4], [5]
- Designing the method to calculate process metrics [4], [5]
- Implementing the designed tools [4], [5]
- Producing the result of the machine learning techniques [4], [5]
- Analyzing the result of the study [4], [5]

3. A Public Dataset of JavaScript Bugs

- Collecting and analyzing JavaScript projects for the benchmark [6], [7], [8]
- Selecting suitable bugs and extracting related bug-fixes [6], [7], [8]
- Analyzing the relationship between the bug taxonomy and the bug-fixing patterns [7]

These results cannot be used to obtain an academic research degree, other than the submitted PhD thesis of **Péter Gyimesi**.

In the PhD dissertation of **Péter Gyimesi** entitled *A Novel Bug Prediction Dataset, Process Metrics and A Public Dataset of JavaScript Bugs*, **Péter Gyimesi** and the corresponding coauthors share the following joint and undividable contributions:

1. A Novel Bug Prediction Dataset and its Validation

- Reviewing the existing related studies [1], [2], [3]
- Defining criteria for the inclusion of projects [1], [2], [3]
- Producing the result of the machine learning techniques [1], [2], [3]

3. A Public Dataset of JavaScript Bugs

- Designing and implementing the framework and the benchmark [6], [7], [8]
- Manually validating the bugs selected for the benchmark [6], [7], [8]
- Extracting and analyzing the bug-fixing patterns [6], [7]
- Creating and validating the bug taxonomy [7]

Referenced publications:

[1] **Péter Gyimesi**, Gábor Gyimesi, Zoltán Tóth, and Rudolf Ferenc. Characterization of Source Code Defects by Data Mining Conducted on GitHub. In 15th International Conference on Computational Science and Its Applications (ICCSA 2015), Banff, AB, Canada, June 22–25, pages 47–62, LNCS, Volume 9159. Springer International Publishing, 2015.

[2] Zoltán Tóth, **Péter Gyimesi**, and Rudolf Ferenc. A Public Bug Database of GitHub Projects and its Application in Bug Prediction. In 16th International Conference on Computational Science and Its Applications (ICCSA 2016), Beijing, China, July 4–7, pages 625–638, LNCS, Volume 9789. Springer International Publishing, 2016.

[3] Rudolf Ferenc, **Péter Gyimesi**, Gábor Gyimesi, Zoltán Tóth, and Tibor Gyimóthy. An Automatically Created Novel Bug Dataset and its Validation in Bug Prediction. Journal of Systems and Software, 2020, 169: 110691.

[4] **Péter Gyimesi**. Automatic Calculation of Process Metrics and their Bug Prediction Capabilities. In Acta Cybernetica, pages 537–559, Volume 23, No 2, 2017

[5] **Péter Gyimesi**. An open-source solution for automatic bug database creation. In Proceedings of the 10th International Conference on Applied Informatics (ICAI 2017), Eger, Hungary, January 30—February 1, pages 111—119, 2017.

[6] **Péter Gyimesi**, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc, and Ali Mesbah. BugsJS: A Benchmark of JavaScript Bugs. In 12th IEEE Conference on Software Testing, Validation and Verification (IEEE ICST 2019), Xi'an, China, April 22–27, pages 90–101, IEEE, Volume 1. IEEE Computer Society Press, 2019.

[7] **Péter Gyimesi**, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc, and Ali Mesbah. BugsJS: A Benchmark and Taxonomy of JavaScript Bugs. Journal of Software Testing, Verification and Reliability (STVR 2021), John Wiley & Sons Publishing

[8] Béla Vancsics, **Péter Gyimesi**, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc, and Ali Mesbah. Poster: Supporting JavaScript Experimentation with BugsJS. In 12th IEEE Conference on Software Testing, Validation and Verification (IEEE ICST 2019), Poster Track, Xi'an, China, April 22–27, pages 375–378, IEEE, Volume 1. IEEE Computer Society Press, 2019.

Szeged, 20 November 2023

Péter Gyimesi

Dr. habil. Rudolf Ferenc

The head of the Doctoral School of Computer Science declares that the declaration above was sent to all of the coauthors and none of them raised any objections against it.

Szeged, 20 November 2023

SZEGEDI TUDOMÁNYEGYETEM
INFORMATIKA DOKTORI ISKOLA
SZEGED

Dr. Márk Jelasity