Új hiba adatbázisok és egy módszertan a karbantarthatóság mérésére RPG hagyaték rendszerekben

Tóth Zoltán

Szoftverfejlesztés Tanszék
Szegedi Tudományegyetem

Szeged, 2019

Témavezető:

Dr. Ferenc Rudolf

Ph.D. értekezés tézisei

Szegedi Tudományegyetem
Informatika Doktori Iskola

Bevezetés

A szoftver rendszerek egyre komplexebbé váltak az évek során. A komplexitás leküzdése érdekében sikeresen alkalmaztunk egyre magasabb absztrakciós szinteket. Mivel a szoftverfejlesztés folyamatának szerves részét képezik az emberek, így a szoftverek fognak hibákat tartalmazni. A XXI. század elején a szoftverhibák közel 60 milliárd dollárnyi kárt okoztak az Amerikai Egyesült Államoknak. Ez a szám 2016-ban 1,1 billió dollár. Az ipari szektorban a határidők általi nyomás sokszor a szoftver minőségének romlásával jár. A 2019-es Boeing botrányban a repülőgépek MCAS rendszerének szoftvere volt hibás, mely automatikusan állítja be a gép röppályáját. A balesetek elkerülhetőek lettek volna, ha legalább a pilótáknak írt kézikönyvben feltüntetik, hogy mi a teendő az adott hiba esetében, azonban kétség kívül a legnagyobb gondot maga a szoftver hibája jelentette, mely 356 ártatlan életet követelt.

Minél későbbi fázisban javítunk egy hibát, az annál többe kerül. Nyilvánvalóan akkor járunk a legjobban, ha a hibát még a tervezési fázisban felfedezzük és javítjuk. Később, a tesztelési fázisban, is hatékonyan megtalálhatjuk a szoftverben megbúvó hibákat, azonban ez már magasabb költséggel jár. Fontos, hogy a kölcségek alacsonyabb szinten tartásához automatizált teszteket is vessünk be. Jelen disszertáció azt tűzte ki céljául, hogy segítse a jövőbeli hibák megtalálását, melyhez felhasználja a múltbéli hibák tapasztalatait, illetve a karbantrathatóság kifinomultabb mérésvel is foglalkozunk, mely kulcsfontosságú tényezője a szoftverek minőségnek.

Ennek fényében a disszertáció két tézispontot fogalmaz meg: *új hiba adatbázisok megalkotása és azok kiértékelése hiba előrejelzéshez*, illetve *egy módszertan a karbantarthatóság mérésére RPG hagyaték rendszerekben*. Minkét témakör a szoftver minőség fontosságát hangsúlyozza, melyek segítségével korszerű megoldásokat bizosítunk a szoftver hibák elleni harcban.

A *publikus hiba adatbázisok* régóta jelen vannak. Annak ellenére, hogy a hiba előrejelzéshez kapcsolódó tanulmányok száma rohamosan nőtt az elmúlt években, csak igen kevés hiba adatbázis érhető el. A hiba adatbázisokat általában nyílt forráskódú projektekből alkotják, melyekben a hibák menedzselése megfelelő hiba követő rednszerrel történt. Ilyen módon, a hibákhoz hozzá tudjuk rendelni az általa érintett forráskód elemeket [10]. Az adatbázisokat általában fájl vagy osztály szinten készítik el. Ahhoz, hogy a hibákat leírjuk, valamilyen jellemzőket kell biztosítani az adatbázis bejegyzéseihez. Ennek egy elhetséges módja a statikus forráskód metrikákkal történik. A létező adatbázisok hiányosságának pótlása érdekében létrehoztunk egy korszerű hiba adatbázist, a *GitHub hiba adatbázist*, mely több mint 50 statikus forráskód metrikát használ a hibák jellemzéséhez. Ezenfelül, az elérhető publikus adatbázisokból létrehoztunk egy nagy *egységes hiba adatbázist*. Az egységes hiba adatbázis létrehozása közben megvizsgáltuk a különböző adatbázisok inkonzisztenciáit, továbbá tanulmányoztuk a metrika halmazok különbözőségét is, melyek az eltérő statikus elemzők használatából fakadtak. Sikerült demonstrálnunk az épített adatbázisok hiba előrejelző képességét is.

A *karbantarthatóság mérése RPG környezetben* egy szűkebb kutatási területet képez. A banki szektorban manapság is rengeteg helyen használnak még IBM nagyszámítógépeket, melyek nagy része emiatt az RPG programozási nyelv használatát is magában foglalja. Az itt használt szoftverekben megbúvó hibák megtalálása és kijavítása akár nemzetbiztonsági kérdés is lehet. Ezzel szemben a kutatási trendek nem tükrözik ennek fontosságát, így kevés a minőség folyamatos ellenőrzése alkalmas eszköz ezen a területen. Más domainben meglévő módszereket ugyan át lehetne ültetni erre a környezetre is, azonban a módszerekhez szükséges esezközök sem állnak rednelkezésre ezen a területen. Utóbbi miatt nem csak egy módszertant, hanem a hozzá tartozó eszközöket is biztosítjuk a korlátok áttörése érdekében.

A disszertáció két fő szekciót tartalmaz, melyek egyben a tézispontokat is alkotják. Jelen tézisfüzetben összegezzük a tézispontokban elért eredményeket.

I. Új adatbázisok hiba előrejelző modellek építéséhez

A tézispontban a Java környezetű publikus hiba adatbázisokkal, illetve azok kiterjesztésével és hiba előrejelzéshez való felhasználásukkal foglalkozunk.

Új publikus hiba adatbázis és kiértékelése a hiba előrejelzésben

Ennek a kutatásnak a középpontjában a létező – statikus forráskód metrikákat használó – publikus hiba adatbázisok gyenge pontjainak feltérképezése állt, illetve ezek alapján egy új hiba adatbázis megalkotása, mely a gyenge pontokat vagy hiányosságot foltozza be. Annak ellenére, hogy a nyílt forráskódú rendszerek közkedvelt hoszt platformja a GitHub, egyetlen egy létező hiba adatbázis sem használja azt forrásként. Az elérhető adatbázisok igencsak korosnak számítanak, így az általuk tartalmazott projektet is idősek. Továbbá ezek az adatbázisok gyakran csak egy szűk forráskód metrikahalmazzal dolgoznak. Az észrevételeinkre alapozva elkészítettünk egy új hiba adatbázist, mely 15 GitHub-os rendszert foglal magában. A rendszerek összesen 3,5 millió sornyi kódot, több mint 114 000 változtatást (commit-ot) és több mint 8 700 lezárt hibát tartalmaznak. A hibákat leíró adatokat automatizált úton állítottuk elő. Az egyes projektekhez körülbelül 6 hónap hosszú időintervallumokat alkottunk (összesen 105-öt), melyek kezdő- és végpontjai egy-egy hivatalos kiadással esnek egybe, és a hiba információkat ezeken az intervallumokon akkumuláltan gyűjtöttük össze. A létező adatbázisokhoz képest fontos különbség, hogy a hiba adatokat nem a hiba kijavítását követő verzióhoz propagáltuk, hanem a megelőző kiadási verzióhoz. Az így előállított adatbázis mind osztály és fájl szinten nyújt információkat. Az elkészült hiba adatbázis mellett – mely a fő kontribúciónk is egyben – két kutatási kérdésre (research question - RQ) kerestük a választ, melyek a következőek:

RQ 1: *Az elkészült GitHub hiba adatbázis alkalmas-e hiba előrejelzéshez? Amennyiben igen, akkor mely gépi tanulási algoritmusok vagy algoritmus családok teljesítenek a legjobban ezen az új adatbázison?*

RQ 2: *Melyik gépi tanulási algoritmusok teljesítenek a legjobban a hibák lefedettsége szempontjából?*

Az első kérdés megválaszolásához 13 különböző hiba előrejelző modellt alkottunk meg a népszerű Weka gépi tanulási keretrendszer segítségével. A tanítási fázis során 10-szeres kereszt validációt, illetve véletlenszerű alulmintavételezést alkalmaztunk [4]. Utóbbit a hibás és nem hibás elemek számbeli különbségének kiegyenlítése végett vetettük be, hogy a tanítás során ugyanannyi hibás és nem hibás elemet mutassunk a rendszernek (a gépi tanuló algoritmusnak). A véletlen alulmintavételezést minden projekt, minden intervallumára 10-szer végeztük el és átlagoltuk az eredményeket. Az öt legjobban teljesítő algoritmust az 1. táblázat mutatja be. A hiba előrejelzésben a Random Forest algoritmus teljesített a legjobban, melyet a magas F-measure értékek támasztanak alá. Osztály szinten átlagosan 0,77-es F-measure értéket sikerült elérni. Fájl szinten hasonló, ámbár kicsit alacsonyabb (0,71-es) átlagos értékek mutatkoztak meg, melynek fő okozója a fájl szintű szűkebb metrikahalmaz. Az elért eredmények pozitív csengésű választ adnak az első kérdésünkre. Kijelenhetjük, hogy a GitHub hiba adatábzis egy jó jelölt lehet a hibák előrejelzéséhez.

Ahhoz, hogy a második kutatási kérdésünket megválaszoljuk, felhasználtuk a véletlen alulmintavételezett bemenetet használó 10 hiba előrejelző modellt, és rajtuk a teljes hiba adatbázist értékeltük ki. A kiértékelést többségi szavazás alapján végeztük, azaz amennyiben egy elemre a 10 modell közül legalább 6 azt jósolja, hogy hibás, akkor az elemet hibásnak ítéltük, különben

1. táblázat. F-measure értékek osztály szinten

Rendszer	SGD	Simple Logistic	SMO	PART	Random Forest
Android Universal I. L.	0.6258	0.5794	0.5435	0.6188	0.7474
ANTLR v4	0.7586	0.7234	0.7379	0.7104	0.8066
Broadleaf Commerce	0.8019	0.8084	0.8081	0.7813	**0.8210**
Eclipse p. for Ceylon	0.6891	0.7078	0.6876	0.7283	0.7503
Elasticsearch	0.7197	0.7304	0.7070	0.7171	0.7755
Hazelcast	0.7128	0.7189	0.6965	0.7267	0.7659
jUnit	0.7506	0.7649	0.7560	0.7262	0.7939
MapDB	0.7352	0.7667	0.7332	0.7421	0.7773
mcMMO	0.7192	0.6987	0.7203	0.6958	0.7418
Mission Control T.	0.7819	0.7355	0.7863	0.6862	**0.8161**
Neo4j	0.6911	0.7156	0.6835	0.6731	0.6767
Netty	0.7295	0.7437	0.7066	0.7521	0.7937
OrientDB	0.7485	0.7359	0.7310	0.7194	0.7823
Oryx	0.8012	0.7842	**0.8109**	0.7754	0.8059
Titan	0.7540	0.7558	0.7632	0.7301	0.7830
Átlag	**0.7346**	**0.7312**	**0.7248**	**0.7189**	**0.7758**

nem hibásnak. A hiba lefedettség számításánál fontos megjegyezni, hogy amennyiben megtalálunk legalább egy a hibát érintő forráskód elemet, akkor a hibát lefedettnek minősítjük. Közel tökéletes lefedettséget tudtunk elérni osztály szinten a Random Forest és a Random Tree algoritmusokkal úgy, hogy a forráskód elemek mindössze 31%-át jelöltük meg hibásként. Fájl szinten is hasonló eredményeket kaptunk, igaz, ott kicsivel több elemet kell megjelölnünk a tökéletes hiba lefedettség eléréséhez. Amennyiben a precision fontosabb a recall metrikánál, akkor jó választás lehet a Naive Bayes algoritmus.

A GitHub hiba adatbázis letölthető a következő URL-ről: `http://www.inf.u-szeged.hu/~ferenc/papers/GitHubBugDataSet/`

Egységesített hiba adatbázis és annak felhasználása hiba előrejelzéshez

A létező hiba adatbázisok feltérképezése közben megfogalmazódott bennünk egy igény, miszerint szükség volna egy egységesített hiba adatbázisra, annak érdekében, hogy a hiba előrejelző módszereket egy sokkal általánosabb, sokrétűbb adathalmazon értékelhessük ki. Így egy kimerítő szakirodalom kutatásba kezdtünk, hogy az összes lehetséges hiba adatbázist felderítsük. 5 jelöltet sikerült találnunk, melyek között a saját GitHub hiba adatbázisunk is megtalálható a PROMISE, az Eclipse, a Bug Prediction és a Bugcatchers hiba adatbázisai mellett. Ezeket az adatbázisokat fésültük össze egy nagy és egységes hiba adatbázisba. Az OpenStaticAnalyzer (OSA) nyílt forráskódú statikus elemzőt használtuk, hogy a több, mint 50 statikus metrikát kinyerjük a különböző rendszerekre és ezekkel jellemezzük az adatbázisban található bejegyzéseket. A létrejövő hiba adatbázisban, így a kiszámított metrikák halmaza egységes lett, ugyanakkor a forráskód elemekhez meghagytuk az eredeti hibaszámokat (az eredeti elemeket az új elemzés eredményeivel a standard Java osztály nevek alapján, míg a fájlokat egyszerűen a nevük alapján párosítottuk). A forráskód elemek párosítása közben fellépő anomáliákat manuális ellenőriztük az okok felderítéséhez. Az egyik fő ok az olyan elemek jelenléte az eredeti adatbázisokban, melyek nem is igazi Java forráskódot tartalmazó állományok (Scala forrásfájlok, package-info fájlok). Néhány esetben az eredeti adatbázisban található rendszerekhez nem tudtuk a megfelelő forráskódot elérni, így

egy másik, de közeli verziót kellett felhasználnunk a párosításban. 624 osztály és 28 fájl szintű adatbázis bejegyzést nem tudtunk párosítani az új megfelelőjükkel. Az arányok végett, 48 242 osztály és 43 722 fájl szintű bejegyzés volt megtalálható az eredeti adatbázisokban. Ez összességében azt jelenti, hogy az elemek 0,71%-át (652 a 92 014-ből) kellett kihagynunk az egységesített hiba adatbázisból.

Az egységesítés folyamata közben megvizsgáltuk az eredeti és az új metrika halmazok közötti átfedéseket (a közös metrikákat), hogy képet kapjunk azok különbözőségeiről. Feltártuk a különböző nevezéktani, illetve definíciós eltéréseket is egyaránt. Szignifikáns eltéréseket találtunk a legegyszerűbb metrikák között is – mint amilyen például a logikai kódsorok száma (LLOC) – mely az elemzőeszközök eltéréséből fakad, ugyanis egyes eszközök forráskódon dolgoznak, míg mások a bájt kódot használják fel bemenetként.

Az adatbázisokat megvizsgáltuk a meta adatok szempontjából is, illetve funkcionális követelmények alapján is. A meta adatok tartalmazzák a használt statikus elemzőt, a granularitást (osztály vagy fájl szintű elemeket tartalmaz az adatbázis), a használt hiba- és verziókövető rendszert, illetve a számított forráskód metrikák halmazát. Egy funkcionális követelményként összehasonlítottuk az eredeti és az új metrikák hiba előrejelző képességét, illetve azt is megvizsgáltuk, hogy ehhez képest együttesen hogyan teljesítenek. A vizsgálathoz a J48 döntési fát (a C4.5 döntési fa egy implementációja) használtuk a Weka gépi tanulási keretrendszerből. A hiba előrejelző modelleket külön-külön tanítottuk az egyes projekteken, melynek sorány 10-szeres keresztvalidációt alkalmaztunk. 0,892-es és 0,886-os F-measure értékeket értünk el osztály szinten, rendre az eredeti metrikák és az OSA metrikák által. Fájl szinten az értékek kicsit alacsonyabbak, azonban az eredeti és az OSA metrikák által kapott eredmények eltérése ott sem számottevő. A két metrikahalmaz együttes alkalmazása az osztály szintű tanításban hozott egy csekély javítást, de fájl szinten ugyanez már nem mondható el. Az alacsonyabb értékeket könnyen okozhatják az eredeti és az új metrikák közötti inkonzisztenciák (például kétféle LLOC metrika, különböző értékekkel). Fontos megjegyeznünk, hogy az alapvetően magasabb F-measure értékek, annak köszönhetőek, hogy a kiértékelésben nem használtunk véletlenszerű alulmintavételezést, mivel csak az eredeti és az új metrikahalmazok hiba előrejelző képességének egymáshoz való viszonya érdekelt minket.

Egy további funkcionális követelményként különböző rendszereket használtunk fel tanításhoz, mind pedig kiértékeléshez, melyet kereszt projektes tanításnak nevezünk. A hiba adatbázisban szereplő összes projekt párosításra elvégeztük ezt a kísérletet. Az eredmények közül a GitHub hiba adatbázis projektjeire vett részhalmaz a leginkább konzisztens, amit a 2. táblázat mutat be. A táblázatban egy elem minél sötétebb szürke, annál magasabb az F-measure értéke.

Összességében a kísérletek megmutatták, hogy az egységes hiba adatbázis hatékonyan használható hiba előrejelzéshez, melyet az elért magas F-measure értékek (0,8 és a fölötti) támasztanak alá. Arra bíztatjuk a terület kutatóit, hogy új módszereik kipróbálásához először próbálják meg alkalmazni a nagy egységesített hiba adatbázisunkat. Ezen felül, új adatbázisokat is örömmel bővítjük az egységes hiba adatbázist. Az egységes hiba adatbázis elérhetősége: `http://www.inf.u-szeged.hu/~ferenc/papers/UnifiedBugDataSet`.

A szerző hozzájárulása

A disszertáció szerzője tervezte meg a GitHub-ról történő hiba adatok kinyerésére irányuló módszert, illetve az ezen adatokból történő GitHub hiba adatbázis felépítésének módszerét. A szerző kutatta fel a publikus hiba adatbázisokkal kapcsolatos szakirodalmat, illetve gyűjtötte össze az adatbázisokhoz tartozó mindennemű információt. A GitHub hiba adatbázisba beválogatni kívánt projektekkel kapcsolatos elvárások megfogalmazásában is aktívan szerepet vállalt. A szerző készítette el a statisztikákat az egységesített hiba adatbázisban szereplő összes projekthez. Ezenfelül,

2. táblázat. Kereszt-projektes tanítás (GitHub – osztály szint)

Tanítás/Kiértékelés	Android Universal I. L.	ANTLR v4	Broadleaf Commerce	Eclipse p. for Ceylon	Elasticsearch	Hazelcast	jUnit	MapDB	mcMMO	Mission Control T.	Neo4j	Netty	OrientDB	Oryx	Titan
Android Universal I. L.	0.943	0.885	0.772	0.822	0.813	0.821	0.827	0.827	0.825	0.836	0.868	0.859	0.852	0.850	0.853
ANTLR v4	0.611	0.929	0.785	0.852	0.843	0.842	0.847	0.847	0.845	0.862	0.893	0.882	0.876	0.874	0.876
Broadleaf Commerce	0.635	0.877	0.934	0.928	0.870	0.859	0.863	0.862	0.860	0.873	0.894	0.886	0.880	0.879	0.879
Eclipse p. for Ceylon	0.611	0.894	0.781	0.867	0.847	0.847	0.851	0.851	0.848	0.864	0.895	0.884	0.879	0.877	0.879
Elasticsearch	0.642	0.868	0.835	0.875	0.922	0.900	0.902	0.900	0.897	0.904	0.914	0.904	0.897	0.895	0.895
Hazelcast	0.635	0.876	0.792	0.831	0.828	0.861	0.864	0.863	0.861	0.871	0.888	0.879	0.872	0.871	0.872
jUnit	0.644	0.849	0.774	0.837	0.819	0.813	0.821	0.822	0.821	0.835	0.867	0.858	0.854	0.853	0.854
MapDB	0.670	0.852	0.799	0.855	0.852	0.853	0.857	0.859	0.858	0.871	0.894	0.884	0.879	0.877	0.878
mcMMO	0.642	0.879	0.793	0.855	0.845	0.849	0.853	0.853	0.853	0.866	0.888	0.880	0.874	0.873	0.875
Mission Control T.	0.611	0.890	0.773	0.843	0.836	0.836	0.840	0.840	0.838	0.856	0.890	0.879	0.872	0.870	0.872
Neo4j	0.611	0.890	0.774	0.843	0.836	0.836	0.841	0.840	0.838	0.856	0.890	0.878	0.871	0.869	0.871
Netty	0.644	0.873	0.787	0.848	0.834	0.831	0.837	0.836	0.834	0.847	0.874	0.876	0.869	0.867	0.868
OrientDB	0.611	0.845	0.800	0.857	0.835	0.841	0.846	0.846	0.845	0.859	0.888	0.878	0.884	0.882	0.882
Oryx	0.712	0.874	0.787	0.845	0.837	0.840	0.845	0.845	0.843	0.857	0.879	0.870	0.865	0.866	0.868
Titan	0.611	0.895	0.787	0.849	0.840	0.843	0.847	0.847	0.845	0.859	0.890	0.880	0.874	0.872	0.878

a statikus forráskód elemzését is ő végezte el az egységes hiba adatbázisban található projektekre. A különböző adatbázisokban használt metrika halmazok összegyűjtése és összehasonlítása, továbbá a meta adatok feltérképezése a szerző saját munkáját képezik. A szerző aktívan részt vett továbbá a hiba előrejelző modellek építésében, mind a GitHub, mind az egységesített hiba adatbázisok esetében. A végső gépi tanulások eredményeit a szerző állította elő, továbbá az azokból levonható következtetéseket is ő fogalmazta meg. A tézisponthoz tartozó alátámasztó publikációk a következőek:

- **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), Beijing, China. Pages 625-638, Published in Lecture Notes in Computer Science (LNCS), Volume 9789, Springer-Verlag. July, 2016.
- Rudolf Ferenc, **Zoltán Tóth**, Gergely Ladányi, István Siket, and Tibor Gyimóthy. A Public Unified Bug Dataset for Java. In Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE'18. Oulu, Finland. Pages 12–21, ACM. October, 2018.

II. Egy módszertan az RPG szoftverrendszerek karbantarthatóságának mérésére

A tézispontban az RPG hatyaték rendszerek karbantarthatóságának mérésével foglalkozunk. Az RPG rendszerek elemzése valós ipari igényen alapszik. Ezen kutatást az R&R Software Kft.-vel közösen vittük véghez, amely cég rendkívül nagy múlttal rendelkezik az RPG rendszerek tervezésében és megvalósításában. A cég vetette fel igényét az RPG nyelvű rendszereik statikus forráskód elemzésére, illetve a rendszerek karbantarthatóságának szofisztikált és megbízható mérésére. A tézispontban található kutatási eredményeket a GOP-1.1.1-11-2012-0323 témaszámú pályázat keretein belül készítettük el.

Szoftverminőség méréshez használható RPG statikus forráskód elemzők összehasonlítása

A kutatás célja, hogy a jelenlegi legkorszerűbb RPG statikus forráskód elemzőket mélyrehatóan és objektíven összehasonlítsa. A kutatási terület a statikus elemzőkből kinyert adatokra fókuszál, amelyeket később magasabb szintű minőségleíró attribútumok meghatározására használhatunk. A SourceMeter egy saját fejlesztésű parancssori eszköz, mely a forráskód különböző tulajdonságait adja eredményül, mint például statikus forráskód metrikákat, kódolási szabálysértéseket és kód duplikációkat. A SonarQube egy minőség menedzsment platform, mely rendelkezik RPG nyelvi támogatással is. Az objektív összehasonlítás megkönnyítése érdekében, a *SourceMeter for RPG* SonarQube-os beépülő modulját (plugin) használtuk, amely kiterjeszti a SonarQube funkcionalitását a SourceMeter képességeivel. Ilyen módon a vizsgált rendszerek egy közös interfésszel rendelkeztek, amely jelentősen megkönnyítette az összehasonlítás elvégzését. Az összehasonlítás bementét 179 rendszer képezte. Az összehasonlításban szereplő szempontok között megtalálható az elemzés sikeressége és mélysége, a forráskód metrikák halmaza, a támogatott kódolási szabálysértések halmaza, illetve a kód duplikációk felfedezésének képessége. A SourceMeter eszköz gond nélkül elemezte az összes forráskódot, míg a SonarQube esetében 3 fájlt nem sikerült a rendszernek analizálni, mivel azok nem támogatott nyelvi konstrukciókat tartalmaztak (például free-form blokkot). A SourceMeter 4 szinten szolgáltatja az eredményeket: *rendszer, program (fájl), procedúra* és *szubrutin* szinten. A SonarQube ehhez képest csak *rendszer* és *fájl* szinteket támogat. A SonarQube limitált mennyiségű statikus forráskód metrikát szolgáltat a SourceMeterhez képest, mely utóbbi a metrikákat alacsonyabb szinteken is számolja (szubrutin és procedúra). A közösen támogatott szabályok halmaza nagy, továbbá mindkét eszköz biztosít olyan kódolási szabályokat, melyeket a másik nem. A SourceMeter eszköz rendelkezik metrika-alapú szabályokkal is, melyek alapján szabálysértést generál a rendszer, ha egy adott forráskód elemhez tartozó metrika értéke kisebb vagy nagyobb, mint az engedélyezett intervallum alsó vagy felső végpontja. A SonarQube eszköz képes az 1-es típusú kód klónok felderítésére, melyek másolás és beillesztés technikával jöttek létre. A SourceMeter ugyanakkor egy absztrakt szintaxis fát (Abstract Syntax Tree - AST) épít, mely a klón detektálás bemenetét képezi, így az képes a 2-es típusú klónok felderítésére is (például a különböző változónevek használatával nem lehet megkerülni a klón megtalálását). Az eszközök kiértékelése után, az alacsony szintű karakterisztikák magasabb szintű jellemzőkre gyakorolt hatását vizsgáltuk meg a platformon, azaz a minőségi indexeket vettük górcső alá. A *technical dept* és a *SQALE* metikákat vettük figyelembe, melyeket a SonarQube szolgáltat. Ezen jellemzők erősen építenek a szabálysértésekre, viszont más alacsony szintű jellemzőket (a forráskód metrikákat és a kód duplikációkat) nem vesznek figyelembe, így ezek a jellemzők nem képesek megfelelő módon tükrözni a rendszer teljes minőségét. Az alacsony szin-

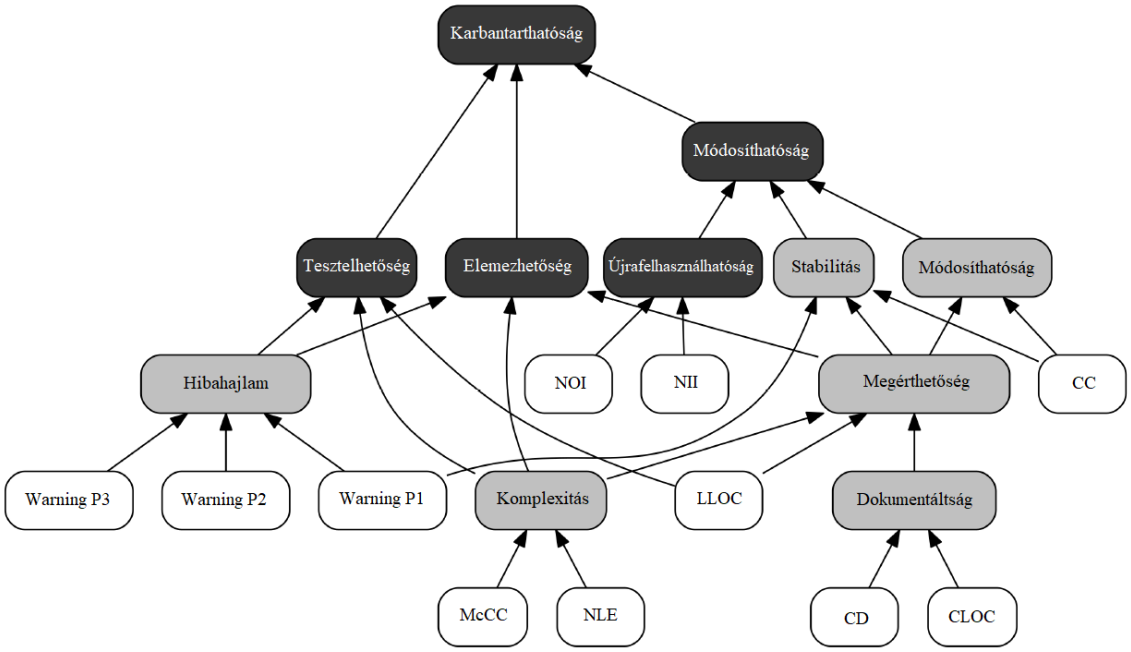
tű jellemzők összehasonlításának összegzése a 3. táblázatban található. A SourceMeter eszközt találtuk fejlettebbnek az elemzés mélységében, a metrikák és a kód klónok tekintetében, míg a kódolási szabályok és az elemzés sikerességét tekintve teljesítményük közel azonos. Ezen faktorokat tekintve a SourceMeter eszközt választottuk ki arra a célra, hogy alacsony szintű jellemzőket biztosítson a szofisztikáltabb, magasabb szintű tulajdonságok meghatározásához.

3. táblázat. Az összehasonlítás összesített eredménye

Szempont	**Eredmény**	**Megjegyzés**
Elemzés sikeressége	Kiegyensúlyozott	A SonarQube néhány fájlt nem tudott elemezni
Elemzés mélysége	SourceMeter	A SourceMeter alacsonyabb szinten is szolgáltat eredményeket
Metrikák	SourceMeter	A SourceMeter lényegesen több metrikát biztosít
Kódolási szabályok	Kiegyensúlyozott	Kiegyensúlyozott szabályhalmazok, nagy közös résszel
Kód duplikációk	SourceMeter	A SourceMeter több duplikált kódot fedett fel

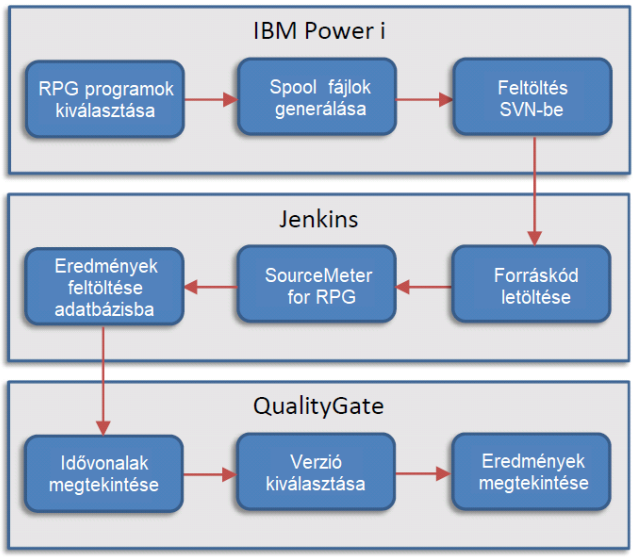
Folyamatos minőség monitorozás integrációja egy meglévő fejlesztési folyamatba – egy esettanulmány

A kutatás célja egy olyan általános és rugalmas minőség modell [1] megalkotása az RPG nyelvre, melyet aztán sikeresen alkalmazhatunk az esettanulmányunkban. Egy jó minőség modell egyet jelent olyan magas szintű tulajdonságokkal, mint amilyenek a minőségi index vagy a karbantarthatósági index, melyek többet árulnak el a rendszerről. A létrehozott minőség modell az ISO/IEC 25010-es szabványon alapszik, melyben a karbantarthatóság, mint kulcs tényező játszik szerepet egy rendszer minőségének meghatározásában (ilyen módon a minőség modell és karbantarthatóság modell kifejezéseket felváltva használjuk a karbantarthatósági modellre, mely valójában csak egy részét képezi a legfelsőbb szintű minőség jellemzőnek). A modell erősen épít



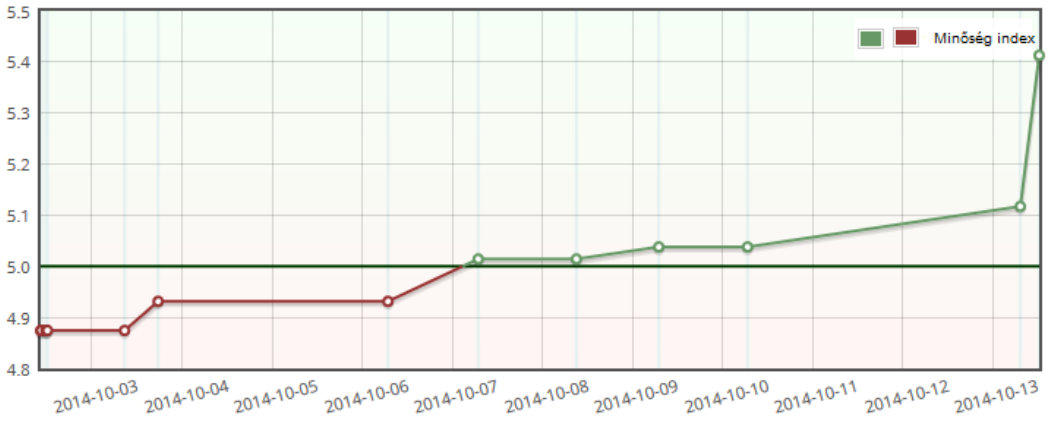
1. ábra. RPG karbantarthatósági modell

az újrafelhasználhatóságára, elemezhetőségére, módosíthatóságára és tesztelhetőségére, mint altulajdonságokra, melyek meghatározzák a rendszer tényleges karbantarthatóságát. A felépített modell a 1 ábrán látható, ahol a sötétszürke csúcsok a karbantarthatóságot alkotó aljellemzők, az általunk definiált világos szürke elemek segítik a szenzor csúcsok (a SourceMeter által kiszámolt alacsony szintű jellemzők) csoportosítását.



2. ábra. A minőség monitorozásának folyamata

A minőség modell definiálása után, kiviteleztünk egy esettanulmányt, melyben a modellt beépítettük egy közepes méretű szoftverfejlesztő cég, az R&R Software Kft., fejlesztési folyamataiba. Az *inicializációs fázisban* a SourceMeter finomhangolása (például a metrika alapú szabályok lehetséges intervallumainak beállítása, tiltott operációk meghatározása) és a minőség modell paraméterezése után (súlyok meghatározása), létrehoztunk egy benchmark-ot, amely négy modult foglal magában.



3. ábra. Karbantarthatóság alakulása

A második, *integrációs fázisban* – melynek lépései a 2. ábrán láthatóak – adaptáltuk a minőség monitorozó módszerünket, mely észrevétlenül épül be a vállalat fejlesztési folyamataiba. Az új változtatások során automatikusan lefut a forráskód elemzés, melynek eredményeit adatbázisban tároljuk. A vállalat egy megadott modul minőségének javítását tűzte ki célul, így egy

refaktorálási fázisban a vállalat néhány fejlesztője csökkentette a rendszerben lévő kritikus és súlyos szabálysértések számát. Ezen tevékenység segítette, hogy a rendszer karbantarthatósága verzióról verzióra növekedhessen, melynek alakulását a 3. ábra mutatja be.

Az *értékelés fázisban* levonhattuk a konklúziót, mely szerint a karbantarthatóság javult a kiválasztott modulon, így elérve a „GO" állapotot, vagyis a megadott minimális karbantarthatósági szintet, amellyel a rendszer már kiadható. A vállalat fejlesztői és menedzserei szerint a módszerünk ipari környeztben történő alkalmazása sikeresnek bizonyult. A vállalaton belül sikeresen tudták ellenőrizni a kódolási konvenciók követését, amely kikényszeríti, hogy a fejlesztők elkerüljenek bizonyos rossz megoldásokat, helyettük bevett technkikákat tanuljanak meg és alkalmazzák azokat a gyakori problémákra. Ezen felül a megközelítésünket kellőképpen rugalmasnak és testre szabhatónak találták (például a benchmark létrehozás és a minőség modell súlyozása gyorsan megtanulható). A fejlesztők szerint a karbantartási munkálatokat hatékonyan tudták elvégezni, melyet a SourceMeter és QualityGate eszközöknek köszönhetnek.

Halstead komplexitás metrikák és a karbantarthatósági index alkalmazása RPG környezetben

A kutatás célja, hogy a minőség modellbe további lehetséges jellemzőket vonjunk be, melyek továbbfejlesztik a modell kifejező képességét. Első lépésként meg kellett határoznunk a Halstead komplexitás metrikák RPG/400 és RPG IV környezetre értelmezhető definícióját. Sajnos nincs standard módja ezen metrikák számításának, mivel a különböző programozási nyelvek különböző nyelvi konstrukciókat tartalmaznak, melyekről sokszor nehéz megállapítani, hogy operandusként avagy operátorként számoljuk őket. RPG esetében egy korábbi definíciós halmaz már létezett az RPG II és RPG III nyelvekre [3], így ezeket terjesztettük ki az RPG/400-ban és az RPG IV-ben bemutatott számos újabb nyelvi konstrukcióra. Következő lépésként megvizsgáltuk a Halstead komplexitás metrikákat, illetve a karbantarthatósági index négy variánsát, hogy ezek milyen korrelációban állnak a többi metrikával (melyek a minőség modellben szerepelnek), továbbá kerestük ezen metrikák alkalmazhatóságát a minőség modell kiterjesztésére, hogy az még jobban le tudja írni a szóban forgó rendszer minőségét. Ehhez a feladathoz főkomponens analízist (Principal Component Analysis - PCA) alkalmaztunk [5]. Az analízis során azt tapasztaltuk, hogy a Halstead komplexitás metrikák egy erősen összefüggő csoportot alkotnak a metrikák halmazán belül, továbbá jól felhasználhatóak az elemzett rendszer jobb leírásához. A PCA kimenetei között szerepel továbbá a faktoranalízisből nyert főkomponensek összetétele, amely megadja, hogy az egyes komponensek a metrikák milyen lineáris kombinációjával jöttek létre. Az 5 legdominánsabb komponens összetételét a 4. táblázat adja meg, mely alapján látható, hogy a Halstead komplexitás metrikák vesznek részt a legnagyobb súllyal az első, legdominánsabb komponens (a variabilitás több mint 50%-át adja) lineráis kombinációjában.

Végső konklúzióként a Halstead Number of Delivered Bugs (HNDB) metrikát javasoljuk belevenni a minőség modellbe, azon belül is a hibahajlam (fault proneness) belső pont gyermekeként, mivel a legnagyobb korrelációs együtthatót a lehetséges hibákkal (warning) produkálta ez a metrika. A HNDB metrika mellett a Halstead Program Vocabulary (HPV) metrikát is ajánlatos belevenni a minőség modell komplexitás belső pontja alá, mivel alacsony korrelációt mutat a McCabe-féle komplexitás metrikával (szubrutinok szintjén), ugyanakkor nagy súllyal szerepel a faktoranalízis lineáris kombinációjának első, azaz legdominánsabb komponnensében. Ilyen módon a rendszer komplexitását a McCabe-féle komplexitás, az NLE (elágazások egymásba ágyazottsága) és a HPV metrikák megfelelő súlyozásával alkotott kombinációja alkotnák, melyek összességében jobban, több szemszögből írnák le a rendszert.

4. táblázat. Faktoranalízis eredménye az 5 legdominánsabb komponensre

	Program					Szubrutin				
	F1	**F2**	**F3**	**F4**	**F5**	**F1**	**F2**	**F3**	**F4**	**F5**
CC	0,153	-0,078	0,616	0,663	0,017	-0,047	-0,081	**0,947**	-0,081	-0,167
HCPL	**0,969**	0,082	-0,090	-0,028	0,072	**0,853**	-0,397	-0,100	-0,193	0,069
HDIF	**0,874**	0,020	0,166	-0,109	-0,249	**0,725**	0,442	0,014	0,291	-0,199
HEFF	**0,891**	0,358	-0,151	-0,009	0,049	**0,774**	-0,198	0,035	0,530	-0,044
HNDB	**0,955**	0,260	-0,070	-0,039	-0,035	**0,928**	-0,052	0,010	0,328	-0,079
HPL	**0,966**	0,230	-0,074	-0,038	-0,007	**0,888**	-0,426	-0,064	-0,026	0,030
HPV	**0,971**	0,022	-0,055	-0,040	0,051	**0,906**	-0,245	-0,102	-0,237	0,046
HTRP	**0,891**	0,358	-0,151	-0,009	0,049	**0,774**	-0,198	0,035	0,530	-0,044
HVOL	**0,956**	0,259	-0,099	-0,030	0,016	**0,827**	-0,508	-0,071	-0,021	0,053
MI	**-0,771**	0,580	-0,136	0,068	-0,127	**-0,892**	-0,285	0,065	0,305	0,032
MIMS	**-0,771**	0,580	-0,136	0,068	-0,127	**-0,891**	-0,286	0,066	0,309	0,032
MISEI	**-0,736**	0,643	0,071	-0,102	0,010	**-0,887**	-0,297	0,076	0,312	0,058
MISM	**-0,745**	0,631	0,049	-0,072	0,034	**-0,877**	-0,313	0,076	0,323	0,061
NLE	0,602	-0,326	0,186	-0,240	-0,288	0,463	0,664	-0,097	0,035	-0,049
McCC	**0,947**	0,260	-0,090	-0,042	-0,020	0,678	0,393	0,092	0,478	0,044
NOI	0,297	-0,430	0,377	-0,422	-0,281	0,258	0,315	0,203	-0,036	**0,782**
CD	0,072	0,215	0,627	-0,516	0,412	**-0,724**	-0,439	0,155	0,348	0,215
CLOC	0,537	0,185	0,455	-0,065	0,150	**0,771**	-0,455	0,008	-0,108	0,247
NII	-	-	-	-	-	-0,059	0,182	0,047	0,208	0,123
LLOC	0,516	-0,502	-0,180	0,185	0,549	**0,899**	-0,393	-0,059	-0,033	0,046
Warning Info	**0,837**	0,206	0,314	0,306	-0,110	0,397	-0,012	**0,897**	-0,088	-0,095
Klón Metrika Szabályok	**0,768**	0,209	0,379	0,362	-0,125	0,188	-0,029	**0,948**	-0,162	-0,129
Komplexitás Metrika Szabályok	**0,899**	0,202	-0,101	-0,068	-0,033	0,538	0,453	0,089	0,311	0,055
Csatolási Metrika Szabályok	**0,813**	0,332	-0,106	-0,008	-0,028	0,156	0,191	0,201	0,028	**0,829**
Dokumentációs Metrika Szabályok	**0,808**	0,126	-0,312	0,053	0,081	0,439	0,125	0,002	0,199	-0,244
Méret Metrika Szabályok	**0,947**	0,021	-0,030	-0,105	0,001	**0,705**	-0,330	-0,011	0,056	0,001

A szerző hozzájárulása

A *SourceMeter for RPG* statikus forráskód elemző kifejlesztése, melynek segítségével RPG/400-as és RPG IV-es rendszereket is elemezhetünk a szerző munkája. Ez az elemző eszköz képezi az egyik legfontosabb alapját az összehasonlításnak, illetve a minőség modellnek is egyaránt. A szerző gyűjtötte össze és dolgozta fel az RPG szoftver rendszerek minőségbiztosításával foglalkozó szakirodalmat. Az elemző eszközök összehasonlításához, illetve a benchmark építéshez használt programokat a szerző gyűjtötte össze. Az összes statikus elemzést a szerző végezte el, illetve azok eredményeit a szerző gyűjtötte össze és dolgozta fel (mind a statikus elemzők összehasonlításánál, mind a minőség modell kiterjesztésénél). Részt vett továbbá az esettanulmány megszervezésében és kivitelezésében. A főkomponens analízis elvégzése és annak eredményei alapján a minőség modell kiterjesztésére vonatkozó javaslatok megfogalmazása a szerző munkáját képezik. A tézisponthoz kapcsolódó alátámasztó publikációk:

- **Zoltán Tóth**, László Vidács, and Rudolf Ferenc. Comparison of Static Analysis Tools for Quality Measurement of RPG Programs. In Proceedings of the 15th International Conference on Computational Science and Its Applications (ICCSA 2015), Banff, Alberta, Canada. Pages 177–192, Published in Lecture Notes in Computer Science (LNCS), Volume 9159, Springer-Verlag. June, 2015.

- Gergely Ladányi, **Zoltán Tóth**, Rudolf Ferenc, and Tibor Keresztesi. A Software Quality Model for RPG. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER). Pages. 91–100. IEEE (2015)

- **Zoltán Tóth**. Applying and Evaluating Halstead's Complexity Metrics and Maintainability Index for RPG. In Proceedings of the 17th International Conference on Computational Science and Its Applications (ICCSA 2017), Trieste, Italy. Pages 575-590, Published in Lecture Notes in Computer Science (LNCS), Volume 10408, Springer-Verlag. July, 2017.

Összefoglalás

A disszertáció tárgyát két fő témakör képezte, melyek az új hiba adatbázisok és azok kiértékelése a hiba előrejelzésben, illetve a karbantarthatóságnak, mint fő minőségi jellemzőnek a mérése RPG hagyaték rendszerekben.

A hiba adatbázisok témakörében összegyűjtöttük az összes eddigi létező adatbázist, amelyek a statikus forráskód metrikákat használják a hibák jellemzéséhez. Ezek az adatbázisok sokszor csupán a klasszikus Chidamber & Kemerer objektum orientált metrikákat használják. Mivel maguk az adatbázisok is sokszor már elég korosak, így a bennük található rendszerek is régiek. A létező adatbázisok a hiba információk kinyerését különböző platformokról végezték, mint például SourceForge, Jira, Bugzilla, CVS és SVN. A GitHub egy jó jelölt lehet a hiba információ beszerzéséhez, mivel a nyílt forráskódú projektek jelentős részét ezen a platformon hosztolják, illetve ezt a platformot nem használták még ilyen célra. Ezen problémák és hiányosságok megoldására létrehoztuk a GitHub hiba adatbázist, mely a kornak megfelelő projekt leíró adatokat foglal magában. Ezen felül megalkottunk egy egységes hiba adatbázist is, melyben így a metrikák halmaza egységes lett. Megmutattuk az egyes adatbázisok heterogenitását az általuk tartalmazott metrikák halmaza szempontjából. Demonstráltuk továbbá mind a Github, mind az egységes hiba adatbázisok hiba előrejelző képességeit is. Véleményünk szerint a kutatóknak először meg kellene próbálkozniuk létező adatbázisok felhasználásával, és csak indokolt esetben létrehozniuk saját, specializált adatbázisokat.

Az RPG rendszerek minőségbiztosítása terén, első lépésként egy kimerítő összehasonlítást végeztünk a korszerű RPG statikus forráskód elemzők között. Ezután a karbantarthatóság, mint a legfontosabb minőségi jellemző, kiszámolására adtunk egy módszertant. Egy közepes méretű vállalat fejlesztési folyamatába sikeresen integráltuk a módszerünket egy esettanulmány keretein belül. Ezt követően megmutattuk, hogy a karbantarthatóság mérését tovább javíthatjuk a Halstead komplexitás metrikák alkalmazásával.
Az 5. táblázat összegzi a disszertáció tézispontjaihoz kapcsolódó alátámasztó publikációkat.

№	[8]	[2]	[9]	[6]	[7]
I.	♦	♦			
II.			♦	♦	♦

5. táblázat. A tézispontokhoz tartozó publikációk

Köszönetnyilvánítás

Valószínűleg senki nem lepődik meg amikor azt mondom, hogy mások segítsége nélkül nehezen jutottam volna el idáig. Először is szeretném megköszönni a témavezetőmnek, Dr. Ferenc Rudolfnak az iránymutatást és azt, hogy megerősített és bíztatott akkor, amikor kétségek között vergődtem. Megmutatta, hogy egy kutatási téma iránti lelkesedés és elhivatottság mindig kifizetődik, ehhez azonban sosem szabad feladni céljainkat. Ebből a gondolkodásmódból talán sikerült valamennyit átültetnem a mindennapi életembe is. Meg szeretném továbbá köszönni, Dr. Gyimóthy Tibornak, a Szoftverfejleszés Tanszék egykori vezetőjének, hogy segítette a PhD tanulmányaim előmenetelét, illetve ellátott engem érdekesbbnél érdekesebb kutatási témákkal. Rengeteg köszönettel tartozom a társszerzőimnek, egyben kollégáimnak, név szerint Dr. Siket

Istvánnak, Gyimesi Péternek, Ladányi Gergelynek és Vidács Lászlónak, akik sokat segítettek a technikai akadályok leküzdésében, továbbá rengeteg jó öltettel szolgáltak. Szeretném megköszönni Szűcs Editnek, hogy stilisztikai és nyelvtani észrevételeivel segítette ennek a disszertációnak a létrejöttét.

Végül, de nem utolsó sorban, szeretném kifejezni hálámat a szeretett családomnak, a first lady-nek, Dorkának és az én egyetlen hercegnőmnek, Olíviának. Nagyra értékelem édesanyám folyamatos bíztatását, illetve a rengeteg segítségét, melyet az évek során nyújtott.

A disszertáció a GINOP-2.3.2-15-2016-00037 azonosítójú, Internet of Living Things című projekt keretében készült, melyet a Magyar Kormány és az Európai Regionális Fejlesztési Alap támogatott.

Tóth Zoltán, 2019

Hivatkozások

[1] Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A probabilistic software quality model. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 243–252. IEEE, 2011.

[2] Rudolf Ferenc, Zoltán Tóth, Gergely Ladányi, István Siket, and Tibor Gyimóthy. A public unified bug dataset for java. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE'18, pages 12–21, Oulu, Finland, Oct 2018. ACM.

[3] Sandra D Hartman. A counting tool for rpg. In *ACM SIGMETRICS Performance Evaluation Review*, volume 11, pages 86–100. ACM, 1982.

[4] Haibo He, Edwardo Garcia, et al. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

[5] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag New York, 2 edition, 2002.

[6] Gergely Ladányi, Zoltán Tóth, Rudolf Ferenc, and Tibor Keresztesi. A software quality model for rpg. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 91–100, Montreal, QC, Canada, March 2015.

[7] Zoltán Tóth. Applying and evaluating halstead's complexity metrics and maintainability index for rpg. In *17th International Conference on Computational Science and Its Applications (ICCSA 2017)*, volume 10408, pages 575–590, Trieste, Italy, July 2017. Lecture Notes in Computer Science (LNCS).

[8] 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)*, volume 9789, pages 625–638, Beijing, China, July 2016. Lecture Notes in Computer Science (LNCS).

[9] Zoltán Tóth, László Vidács, and Rudolf Ferenc. Comparison of static analysis tools for quality measurement of rpg programs. In *15th International Conference on Computational Science and Its Applications (ICCSA 2015)*, volume 9159, pages 177–192, Banff, AB, Canada, June 2015. Lecture Notes in Computer Science (LNCS).

[10] Chadd Williams and Jaime Spacco. Szz revisited: verifying when changes induce fixes. In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 32–36. ACM, 2008.